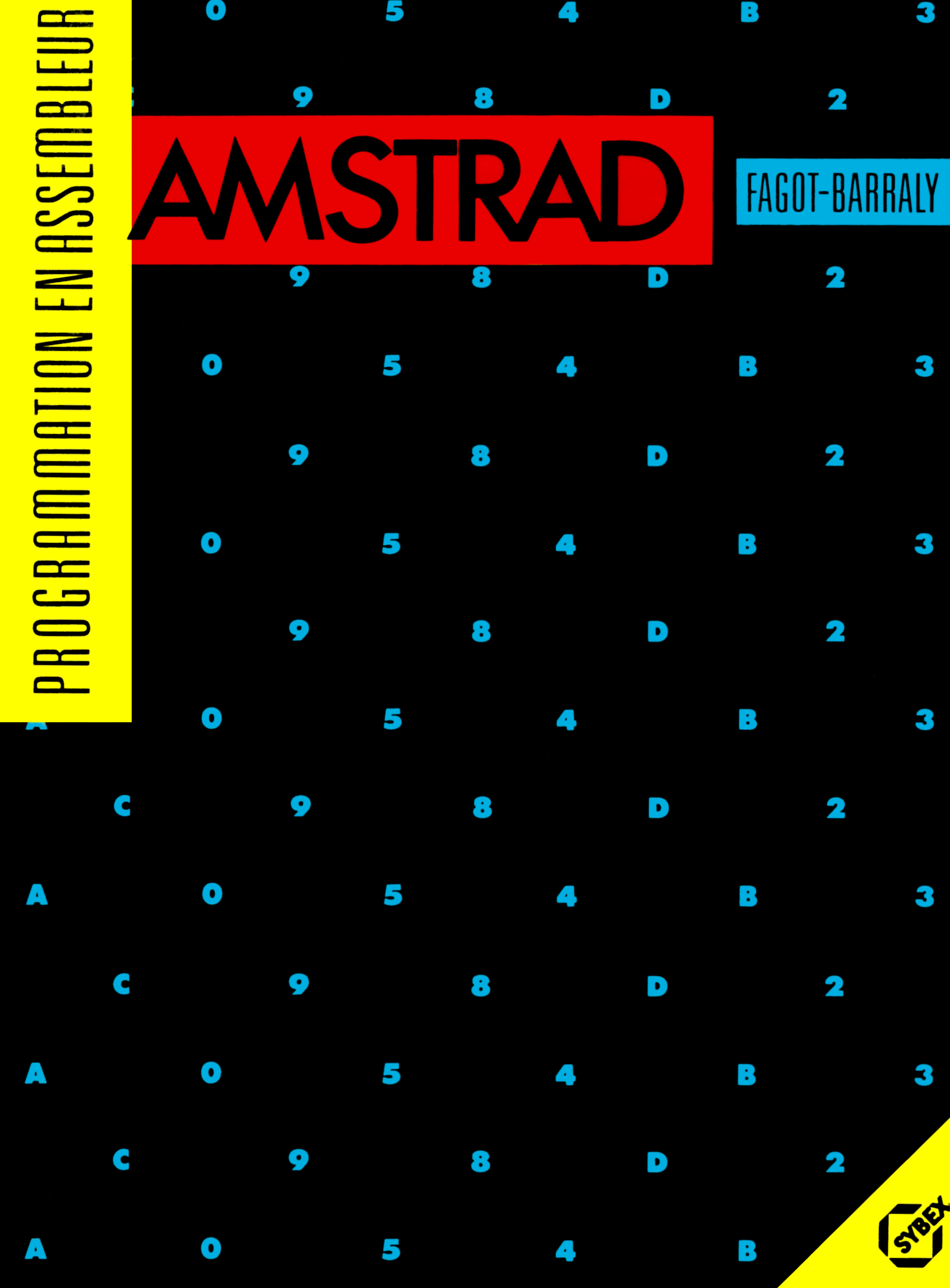

PROGRAMMATION EN ASSEMBLEUR
AMSTRAD
FAGOT-BARRALY
SYBEX





Paris • Berkeley • Düsseldorf • Londres


AMSTRAD
PROGRAMMATION EN ASSEMBLEUR

Tous les efforts ont été faits pour fournir dans ce livre une information complète et exacte. Néanmoins, SYBEX n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

SYBEX n'est lié à aucun constructeur.

Copyright version originale © SYBEX, 1985

Tous droits réservés. Toute reproduction même partielle, par quelque procédé que ce soit, est interdite sans autorisation préalable. Une copie par xérographie, photographie, film, bande magnétique ou autre, constitue une contrefaçon passible des peines prévues par la loi sur la protection des droits d'auteur.

ISBN 2-7361-0136-2

INTRODUCTION

Ce livre est destiné à tous ceux qui connaissent les éléments de base de la programmation BASIC de l'Amstrad et qui souhaitent maintenant franchir l'étape suivante, l'étape du langage machine et de sa forme plus compréhensible : l'assembleur.

Il existe des ouvrages très bien faits qui traitent de ce sujet mais la plupart d'entre eux s'adressent à des lecteurs déjà initiés et ne s'intéressent que très peu à ceux qui font leurs premiers pas dans ce domaine. Aussi avons-nous considéré qu'il y avait place dans le monde du livre pour un ouvrage qui prendrait les débutants par la main et guiderait, avec mille égards, leur entrée dans l'univers fascinant des microprocesseurs.

Nous nous sommes, tout au long de cette étude, fixé la règle suivante : pour chacun des exemples analysés nous ferons apparaître la partie assembleur, sa traduction en langage machine et la façon d'inclure ces codes machine dans un programme BASIC. Face à son ordinateur, le lecteur pourra donc, au fur et à mesure, mettre en pratique les connaissances qu'il viendra d'acquérir.

Le Chapitre 1 donnera les bases indispensables de l'arithmétique binaire car, ne l'oublions pas, un ordinateur ne connaît en réalité pas autre chose que les chiffres 0 et 1.

Le Chapitre 2 rappellera comment est conçue la mémoire écran de l'Amstrad. Cette étude est rendue nécessaire par le fait que la majorité des programmes écrits en langage machine sont des animations de type vidéo. Ajoutons que cela nous permettra d'apprécier, de visu, les résultats d'une bonne partie des programmes assembleur de ce livre. On trouvera aussi dans ce chapitre un programme de démonstration qui nous fera voir la différence flagrante dans les vitesses d'exécution d'un programme BASIC et de son équivalent assembleur.

Le Chapitre 3 nous fera pénétrer au cœur du microprocesseur : c'est un chapitre consacré aux différents registres, registres dont la connaissance est obligatoire pour aborder la suite de ce livre. Nous trouverons aussi dans ce chapitre l'étude des différentes façons d'utiliser une instruction assembleur suivant le mode d'adressage choisi.

Le Chapitre 4 sera consacré à l'étude de notre premier programme écrit en assembleur : les moindres détails seront expliqués. Quelques

pages seront réservées aux lecteurs qui ont à leur disposition la cassette éditeur/assembleur Zen.

Le Chapitre 5 analysera les principales instructions nécessaires à la programmation du microprocesseur de l'Amstrad. De nombreux exemples seront fournis et ceci toujours avec la manière dont le BASIC et le langage machine seront reliés l'un à l'autre. Nous avons enchaîné l'étude des diverses instructions sans nous soucier d'un quelconque ordre logique ou alphabétique : c'est la seule notion de progressivité qui nous a guidés.

Notre principal souhait est d'avoir fait un livre accessible facilement, un livre que l'on ne referme pas au bout de quelques pages devant la supposée trop grande ampleur de la difficulté.

1

L'ARITHMÉTIQUE BINAIRE

La base dix

Le système de numération à base 10 est le système que nous utilisons dans la vie de tous les jours. On le connaît sous le nom de système décimal et c'est le nombre 10 qui y joue le rôle primordial.

Pour commencer notre étude rappelons ce que valent les premières puissances de 10 :

$$10^0 = 1$$
$$10^1 = 10$$
$$10^2 = 10 \times 10 = 100$$
$$10^3 = 10 \times 10 \times 10 = 1000$$

Par convention, n'importe quel nombre avec l'exposant 0 est égal à 1, et 10 n'échappe pas à la règle : $10^0 = 1$.

A l'aide de ces puissances, il est possible d'écrire un quelconque nombre entier.

$$2548 = 2000 + 500 + 40 + 8$$

or
$$2000 = 2 \times 1000 = 2 \times 10^3$$
$$500 = 5 \times 100 = 5 \times 10^2$$
$$40 = 4 \times 10 = 4 \times 10^1$$
$$8 = 8 \times 1 = 8 \times 10^0$$

ce qui donne : $2548 = 2 \times 10^3 + 5 \times 10^2 + 4 \times 10^1 + 8 \times 10^0$.

D'une manière analogue, on aura :

$$4706 = 4000 + 700 + 6$$

soit : $4706 = 4 \times 10^3 + 7 \times 10^2 + 0 \times 10^1 + 6 \times 10^0$.

$10^3 = 1000$	$10^2 = 100$	$10^1 = 10$	$10^0 = 1$
2	5	4	8
4	7	0	6

Naturellement, il nous est loisible de choisir des nombres plus grands car il suffira de prendre des puissances de 10 avec un exposant supérieur.

Rien de bien compliqué dans tout cela. Passons à l'étude d'une autre base mais, auparavant, notons bien quelque chose que nous retrouverons dans tout ce chapitre : les chiffres utilisés en base 10 vont de 0 à 9 ; ils sont tous inférieurs à cette base.

La base cinq

Les puissances de 5 se calculent facilement : $5^0 = 1$; $5^1 = 5$; $5^2 = 25$; $5^3 = 125$. Pour écrire un nombre en base 5, il va falloir constituer un tableau analogue au précédent mais, bien entendu, sa première ligne sera écrite avec les puissances de 5. Soit par exemple à traduire 138 dans le système à base 5 :

$5^3 = 125$	$5^2 = 25$	$5^1 = 5$	$5^0 = 1$
1	0	2	3

On a cherché combien de multiples de 125 (5^3) étaient contenus dans 138 :

1 fois et il reste 13 : $138 = 1 \times 125 + 13$.

Puis on a cherché combien de fois on pouvait faire rentrer 25 (5^2) dans 13 :

0 fois et il reste toujours 13 : $138 = 1 \times 125 + 0 \times 25 + 13$.

Il a fallu alors chercher combien de fois allait rentrer 5 (5^1) dans 13 :

2 fois et il reste 3 : $138 = 1 \times 125 + 0 \times 25 + 2 \times 5 + 3$.

Dernière phase de l'opération : dans le reste qui vaut à ce moment-là 3, combien de fois peut-on faire rentrer 1 (5^0) ?

3 fois et il ne reste rien : $138 = 1 \times 125 + 0 \times 25 + 2 \times 5 + 3 \times 1$.

On a donc en résumé :

$$138 = 1 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 3 \times 5^0$$

et on en déduit que 138 s'écrit 1023 en base 5.

Prenons un deuxième exemple : quelle est la valeur de 279 en base 5 ?

$5^3 = 125$	$5^2 = 25$	$5^1 = 5$	$5^0 = 1$
2	1	0	4

$279 = 2\times125 + 1\times25 + 0\times5 + 4\times1$
ou $279 = 2\times5^3 + 1\times5^2 + 0\times5^1 + 4\times5^0$

Par suite 279 s'écrit 2104 en base 5.

En pratique, pour écrire un nombre décimal dans une autre base, on utilise le plus souvent la méthode dite «des divisions successives».

```
279 | 5
  4 | 55 | 5
       0 | 11 | 5
             1 | 2 | 5
                 2 | 0
```

Elle consiste à diviser le nombre par 5 puis le quotient par 5 puis le nouveau quotient obtenu par 5 et cela jusqu'à ce que le dernier quotient soit nul. Il ne reste plus alors qu'à écrire la liste des différents restes en prenant la précaution essentielle de les copier dans l'ordre inverse. Les restes, dans notre exemple, étant 4,0,1,2 on écrit alors : 279 = 2104 (base 5).

Si nous avons maintenant à traduire en décimal un nombre déjà écrit en base 5, il faudra inscrire ce nombre dans un tableau conçu comme les précédents et ensuite le calculer.

Soit à écrire 3421 (base 5) en base 10.

$5^3 = 125$	$5^2 = 25$	$5^1 = 5$	$5^0 = 1$
3	4	2	1

On en déduit que 3421 (base 5) = $3\times5^3 + 4\times5^2 + 2\times5^1 + 1\times5^0$.
Et l'on obtient : 3421 (base 5) = $3\times125 + 4\times25 + 2\times5 + 1\times1 =$ 486.

Remarquons, pour terminer, que les seuls chiffres utilisés en base 5 sont 0, 1, 2, 3, 4.

Il est conseillé au lecteur de s'assurer, avec quelques exercices dont il aura pris les nombres au hasard, que tout ce qui a été vu est bien assimilé. Non pas que la base 5 ait une quelconque importance en

informatique, mais elle permet de comprendre sans peine les mécanismes des systèmes de numération.

La base deux

Nous arrivons maintenant au cœur du problème : voici le système de numération (dit système binaire) qu'utilisent les ordinateurs.

Tout d'abord, les puissances de 2 : $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$.

Puis maintenant, un exemple : on décide d'écrire 23 en binaire :

$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	0	1	1	1

La plus grande puissance de deux qui rentre dans 23 est 16 (2^4) : le reste est 7. Peut-on ensuite faire rentrer 8 (2^3) dans 7 : la réponse est non et le chiffre 0 a été placé dans la case correspondante.

Par contre 4 (2^2) est contenu dans 7 : on écrit le chiffre 1 dans la troisième case et on note le nouveau reste : 3.

2 (2^1) étant plus petit que 3, on écrit le chiffre 1 dans la quatrième case et puisque le reste vaut alors 1, il nous faut encore écrire 1 mais cette fois-ci dans la dernière colonne.

23 = 10111 (base 2).

Heureusement pour nous, la méthode des divisions successives par 2 va nous donner la réponse d'une manière plus sûre et plus rapide :

```
23 | 2
 1 | 11 | 2
   |  1 | 5 | 2
        | 1 | 2 | 2
              0 | 1 | 2
                | 1 | 0
```

23 = 10111 (2)

Voici d'autres exemples dont les calculs intermédiaires seront laissés à la charge du lecteur :

34 = 100010 (2)
150 = 10010110 (2)
255 = 11111111 (2)

Il reste à voir comment passer de la base 2 à la base 10.

Admettons que l'on veuille écrire 1111011 en décimal. On reconstitue le tableau dans lequel sont indiquées les puissances de 2 et on y écrit notre nombre :

$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
1	1	1	1	0	1	1

On passe plus de temps à faire le tableau qu'à obtenir la réponse !

1111011 = 64 + 32 + 16 + 8 + 2 + 1 = 123 (décimal)

Il est nécessaire de remarquer que, dans ce que nous venons de voir, les seuls chiffres utilisés sont le 0 et le 1, à savoir les chiffres inférieurs à la base.

La base seize

C'est le système (appelé hexadécimal ou hexa) dont les informaticiens ne peuvent se passer, alors qu'au premier abord on pourrait se demander ce que vient faire son étude dans ce livre.

Les 16 chiffres nécessaires à l'écriture dans cette base sont tout d'abord 0,1,2,3,4,5,6,7,8,9...

Mais après le 9, le 10 peut-être ? Mais non, puisque c'est un nombre. Comme il nous manque 6 chiffres, on les a remplacés par les premières lettres de l'alphabet.

Chiffres	A	B	C	D	E	F
Valeurs	10	11	12	13	14	15

Ainsi 12 s'écrit C, 14 s'écrit E.

Là encore, les méthodes de conversion étudiées dans les paragraphes précédents vont s'appliquer.

Soit à écrire 300 en base 16 : les divisions successives doivent se faire par 16.

```
300 |16
  C |18 |16
    |  2|1  |16       300 = 12C (hexa)
        |1  |0
```

Passons à un autre exemple après avoir remarqué que le reste de la première division, qui valait 12, a été remplacé par C.

```
5032 |16
   8 |314 |16
     |   A|19 |16        5032 = 13A8 (hexa)
          |  3|1  |16
              |1  |0
```

Si l'on souhaite traduire en décimal un nombre déjà écrit en base 16, on utilise les puissances de 16.

$16^0 = 1$ $16^1 = 16$ $16^2 = 256$ $16^3 = 4096$

3D4F (hexa) s'écrit $3 \times 16^3 + D \times 16^2 + 4 \times 16^1 + F \times 16^0$
donc 3D4F = $3 \times 4096 + 13 \times 256 + 4 \times 16 + 15$
soit 3D4F = 15695 (base décimale).

Faut-il rappeler aux utilisateurs de l'Amstrad qu'il existe une fonction BASIC, HEX$, qui donne immédiatement la valeur hexadécimale d'un nombre décimal ?

PRINT HEX$ (15695) et l'ordinateur affichera **3D4F**

Il nous faut maintenant comprendre où réside l'intérêt en informatique du système hexadécimal et pour cela comparer les représentations d'un même nombre décimal suivant que l'on veuille l'écrire en base 2 ou en base 16.

183 (décimal) = 1011 0111 (binaire)
183 (décimal) = B 7 (hexa)

On sépare les huit chiffres binaires en deux groupes de quatre :

1011 et 0111

or 1011 = $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8+2+1 = 11$ (décimal)

et 0111 = $0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 4+2+1 = 7$ (décimal)

En remarquant que 11 décimal s'écrit B en hexadécimal, on voit de façon immédiate la correspondance entre les bases 2 et 16. Il est tout à fait possible de passer directement de la base 2 à la base 16 sans avoir à connaître précisément le nombre décimal dont il s'agit.

Essayons encore en partant du nombre décimal 143 qui s'écrit 10001111 en base 2 :

$$\underbrace{1000}_{8}\ \underbrace{1111}_{F} = \text{8F en hexadécimal}$$

Bien entendu, on passera tout aussi facilement de la base 16 à la base 2 en ayant bien à l'esprit le tableau suivant.

Décimal	Binaire	Hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Que peut bien valoir par exemple en binaire le nombre hexadécimal 4A ?

Réponse : 0100 1010
4 A

Et le nombre hexadécimal 37E ?

Réponse : 0011 0111 1110
3 7 E

Dans ce dernier exemple, les deux premiers chiffres 0 sont inutiles et ne servent qu'à la compréhension de la règle qu'il faudra toujours respecter : le partage du nombre binaire doit se faire par groupes de quatre et cela toujours en partant de la droite.

Puisque l'Amstrad dispose de la fonction HEX$, il nous sera possible d'éviter la tâche fastidieuse de conversion d'un nombre en binaire et cela grâce à l'utilisation intermédiaire de la base 16.

Admettons que l'on veuille convertir 1000 (décimal) en binaire

PRINT HEX$ (1000) donne 3E8

On en déduit sans peine le résultat recherché :

0011 1110 1000
3 E 8

Cette réponse pourra naturellement être vérifiée en utilisant la fonction BIN$.

OPÉRATIONS DANS LES BASES 2 ET 16

Nous nous limiterons dans ce paragraphe à l'addition et à la soustraction.

Addition

On considère le mécanisme de l'addition dans notre système décimal et on l'applique à la base 2.

```
  207
+ 321
-----
= 528
```

Dans cet exemple, les chiffres de chaque colonne s'ajoutent : comme on n'atteint jamais 10 (la base, ne pas l'oublier), il n'y a aucun problème. Il va en être de même dans les additions binaires suivantes car les totaux ne dépasseront jamais 2 (valeur de la base binaire) :

```
  101100         100011
+ 010001       + 000100
--------       --------
= 111101       = 100111
```

Reste à voir le cas de la retenue :

```
   1
  447
+ 223
-----
= 670
```

Dans cette addition décimale, 7 et 3 donnent 10, c'est-à-dire très précisément la valeur de la base. On écrit alors 0 au-dessous des chiffres 7 et 3 puis on reporte 1 dans la colonne suivante. Nous procéderons exactement de la même façon avec le système binaire.

```
      1
  10001
+ 01001
-------
= 11010
```

La somme des deux chiffres de droite donne 2 (valeur de la base). Le dernier chiffre du résultat sera donc un 0 et la retenue 1 sera écrite en haut de la colonne suivante. Le reste des calculs s'effectue ensuite sans difficulté.

Essayons encore :

```
     1 1
   100101
 + 000101
 --------
 = 101010
```

Il n'y a rien à redire, passons à un autre exemple :

```
      11
   101011
 + 010011
 --------
 = 111110
```

La somme des deux chiffres de droite donnant 2, on a écrit 0 comme dernier chiffre pour la réponse et on a retenu 1. L'addition de cette retenue avec les deux chiffres 1 de la deuxième colonne donne alors 3, ce qui se traduit par l'écriture du chiffre 1 dans la réponse et la pose d'une retenue en haut de la troisième colonne.

Il faut bien reconnaître que le risque d'erreur n'est pas négligeable lorsque l'on a à effectuer des calculs en binaire. Aussi, une méthode souvent utilisée consiste à traduire les nombres en hexadécimal, à les ajouter alors, puis à reconvertir si nécessaire le résultat en base 2.

Décidons de faire en hexadécimal les additions suivantes :

```
                      1
   34B5             5264
 + 6614             A32E
 ------           ------
 = 9AC9           = F592
```

Si l'on se souvient de la correspondance :

A = 10 B = 11 C = 12 D = 13 E = 14 F = 15

on comprend directement comment la première opération a été faite.

5 + 4 = 9
B + 1 = 11 + 1 = C
4 + 6 = 10 = A
3 + 6 = 9

Pour ce qui concerne la deuxième addition, les choses se décomposent de la manière suivante :

4 + E = 4 + 14 = 18.

La retenue qui correspond à 10 dans notre système habituel est égale à 16 dans le système hexadécimal. Ce qui fait qu'après avoir posé la retenue en haut de la deuxième colonne, il restera 2 à écrire comme chiffre de droite de la réponse, réponse qui se complète ensuite par :

1 + 6 + 2 = 9
2 + 3 = 5
5 + A = 5 + 10 = 15 = F

Autres exemples :

```
  1 1            111
  4BC3           FFFF
+ 2A2F         + FFFF
------         ------
= 75F2         = 1FFFE
```

Nous sommes bien d'accord, n'est-ce pas, dans le système hexadécimal il n'y a de retenue qu'à partir de 16.

Soustraction

Gardons le système précédent de numération et intéressons-nous au calcul d'une différence :

```
  9AE7
- 49B3
------
= 5134
```

C'est, somme toute, plutôt facile à comprendre :

7 − 3 = 4
E − B = 14 − 11 = 3
A − 9 = 10 − 9 = 1
9 − 4 = 5

Alors, essayons les retenues :

```
  9B54
- 6A29
------
= 312B
```

On a tendance à dire 9 ôté de 14, la force de l'habitude nous faisant rajouter une dizaine à 4. En réalité, puisque nous sommes en hexadécimal, ce n'est pas dix que l'on doit ajouter à 4 mais seize. Il s'agit, du coup, de faire 9 ôté de 20 : reste 11 c'est-à-dire B. Naturellement, la retenue ne doit pas être perdue dans la suite des calculs.

5 – 3 (dont 1 de retenue) = 2
B – A = 11 – 10 = 1
9 – 6 = 3

Deux autres exemples :

```
   4A85          ABCD
 – 1F2E        – 2FFF
 ------        ------
 = 2B57        = 7BCE
```

Les utilisateurs de l'Amstrad auront toutes les facilités pour se familiariser avec ce genre d'exercices. Pour faire vérifier par la machine ces deux calculs, il suffira de taper :

```
PRINT HEX$ (&H4A85 – &H1F2E)
```

et

```
PRINT HEX$ (&HABCD – &H2FFF)
```

On en arrive maintenant au calcul de la différence entre deux nombres écrits dans le système binaire. La méthode de soustraction directe peut être employée :

```
   101011
 – 001001
 --------
 = 100010
```

Mais les programmeurs lui préfèrent une autre méthode, celle dite du «complément à deux», car on comprend bien que la soustraction qui vient d'être effectuée aurait été plus compliquée si des retenues étaient apparues.

Le complément à deux

Considérons le nombre décimal 17.

Sa conversion en binaire donne 10001. Pour obtenir le complément à 2 de ce nombre, on respecte les trois étapes suivantes :

- on écrit notre nombre sur huit chiffres en rajoutant des 0 devant :
  00010001
- on remplace chaque 0 par 1 et chaque 1 par 0 :
  11101110
- on ajoute 00000001 à ce résultat :
  11101111

Le nombre que l'on obtient est appelé le complément à 2, sur huit chiffres, de 17, et l'ordinateur considérera que c'est l'opposé de 17, c'est-à-dire le nombre –17. Oui, vous avez bien lu, dans le mode complément à 2, le nombre binaire 11101111 est égal à –17 !

Comment s'en assurer ? En partant de l'idée toute simple qui consiste à dire : puisque, en ajoutant 17 et son opposé –17, on obtient 0, on doit normalement, en ajoutant 00010001 et 11101111, obtenir aussi 0.

Voyons cela :

```
   11111111
    00010001
+   11101111
------------
= (1)00000000
```

Les deux chiffres 1 de la droite font apparaître une retenue que l'on retrouve ensuite de colonne en colonne. Il faut tout de même noter qu'il ne doit pas être tenu compte de la dernière retenue et que nous prendrons l'habitude de la négliger. Nous verrons bientôt que l'ordinateur ne procède pas autrement : pour lui aussi, la dernière retenue de gauche tombe "à l'eau".

Un autre exemple : essayons d'écrire –50 en binaire sous la forme complément à 2 :

00110010	50 décimal
11001101	chiffres inversés
11001110	ajout de 1

donc –50 s'écrit	11001110	en binaire
ou	C E	en hexadécimal

Voici, tels quels, quelques résultats qui doivent permettre au lecteur d'assimiler parfaitement la façon dont l'ordinateur écrit les nombres négatifs :

−5 (décimal) = 11111011 (binaire) = FB (hexa)
−20 (décimal) = 11101100 (binaire) = EC (hexa)
−100 (décimal) = 10011100 (binaire) = 9C (hexa)

Avant de passer à autre chose, revenons quelques minutes sur la façon dont on s'y prendra pour faire une différence binaire maintenant que nous savons utiliser la technique du complément à 2.

Soit à calculer 101000 − 10111.

On cherche l'opposé du deuxième terme de la soustraction en mode complément à 2 : on obtient 11101001.

Il reste alors à ajouter le premier terme avec l'opposé du deuxième :

```
  111 1
    00101000
+   11101001
------------
= (1)00010001
```

La réponse est la suivante : 101000 − 10111 = 10001.

OPÉRATEURS LOGIQUES

En dehors des calculs arithmétiques habituels, on peut effectuer sur les nombres binaires des opérations d'un type spécial que l'on appelle les opérations logiques. Elles ne présentent aucune difficulté car en aucun cas ne se pose le problème des retenues.

Le OU logique

Cette opération respecte les règles suivantes :

```
     0        0        1        1
  OU 0     OU 1     OU 0     OU 1
  ----     ----     ----     ----
   = 0      = 1      = 1      = 1
```

C'est la même chose avec des nombres binaires plus grands :

```
   101101          101000
OU 110101       OU 001100
---------       ---------
 = 111101        = 101100
```

L'Amstrad dispose d'une instruction qui effectue ce type de calculs : c'est le mot clé OR. Demandons-lui quelques résultats :

PRINT 46 OR 100 ; réponse : **110**

```
    101110 <-------- 46
OR 1100100 <-------- 100
----------
=  1101110 <-------- 110
```

PRINT 50 OR 0 ; réponse : **50**

```
   110010 <-------- 50
OR 000000 <-------- 0
---------
=  110010 <-------- 50
```

L'opérateur OR va nous servir en assembleur car il permet de forcer l'un des chiffres binaires à passer à 1. Voyons comment :

PRINT 82 OR 1 ; réponse : **83**

```
   1010010 <-------- 82
OR 0000001 <-------- 1
----------
=  1010011 <-------- 83
```

PRINT 91 OR 1 ; réponse : **91**

```
   1011011 <-------- 91
OR 0000001 <-------- 1
----------
=  1011011 <-------- 91
```

Dans le premier exemple, on part d'un nombre dont le dernier chiffre binaire (bit 0) est égal à 0. Après utilisation de OR 1, ce dernier chiffre a été porté à 1 sans qu'aucun des autres chiffres ait été modifié.

Dans le deuxième cas, on est parti d'un nombre qui se terminait déjà par un 1. OR 1 n'a modifié ni ce chiffre ni naturellement aucun

des autres. On en conclut donc que si l'on effectue OR 1 avec n'importe quel nombre, on aura un résultat dont le dernier chiffre (bit 0) vaudra obligatoirement 1.

D'une façon analogue, en calculant OR 4 avec n'importe quel nombre, on sera certain que le troisième chiffre en partant de la droite est un 1 (bit 2) :

PRINT 19 OR 4 ; réponse : **23**

```
   10011 <——————— 19
OR 00100 <——————— 4
= 10111 <——————— 23
```

Le troisième chiffre est bien passé à 1.

PRINT 52 OR 4 ; réponse : **52**

```
   110100 <——————— 52
OR 000100 <——————— 4
= 110100 <——————— 52
```

Le troisième chiffre est resté à 1.

Le ET logique

Le ET logique est défini par les règles suivantes :

0	0	1	1
ET 0	ET 1	ET 0	ET 1
= 0	= 0	= 0	= 1

Quelques exemples :

101100	101000
ET 011001	ET 110111
= 001000	= 100000

On peut faire faire ces calculs par l'ordinateur et cette fois, c'est le mot réservé AND qui va nous servir.

PRINT 30 AND 40 ; réponse : **8**

```
    11110  <------  30
AND 101000 <------  40
-----------
=   001000 <------  8
```

On retrouve l'instruction AND en assembleur car, grâce à elle, nous pouvons mettre à 0 n'importe quel chiffre binaire. Supposons que nous ayons un nombre et que nous voulions forcer à 0 son chiffre de droite (bit 0). On utilisera AND 254 et voici pourquoi :

PRINT 201 AND 254 ; réponse : **200**

```
    11001001 <------  201
AND 11111110 <------  254
-------------
=   11001000 <------  200
```

Seul le dernier chiffre a été mis à 0, les autres sont restés les mêmes. 254 a en effet la particularité d'être constitué de sept chiffres 1 suivis d'un seul 0.

Si nous étions partis d'un nombre se terminant déjà par 0, AND 254 n'aurait rien modifié, ce qui nous permet de donner la conclusion suivante : quel que soit le nombre considéré, en le combinant avec 254 on pourra être assuré qu'il se terminera par 0.

Il est possible d'annuler n'importe quel chiffre d'un nombre avec l'opérateur AND. AND 124, par exemple, annulera le chiffre de gauche (bit 7) mais en même temps les deux chiffres de droite (bits 0 et 1) de n'importe quel nombre de huit chiffres.

PRINT 245 AND 124 ; réponse : **116**

```
    11110101 <------  245
AND 01111100 <------  124
-------------
=   01110100 <------  116
```

Le OU exclusif logique

Noté XOR, le OU exclusif obéit aux mêmes règles que le OU déjà défini sauf pour la quatrième partie :

```
   0          0          1          1
XOR 0      XOR 1      XOR 0      XOR 1
-----      -----      -----      -----
 = 0        = 1        = 1        = 0
```

Le résultat n'est égal à 1 que lorsqu'un des chiffres et un seulement est égal à 1.

Tapons au clavier de notre ordinateur :

PRINT 30 XOR 40 ; réponse : **54**

```
      11110 <---------- 30
XOR 101000 <---------- 40
----------
=   110110 <---------- 54
```

PRINT 25 XOR 100 ; réponse : **125**

```
      11001 <---------- 25
XOR 1100100 <---------- 100
-----------
=   1111101 <---------- 125
```

L'opérateur XOR est mis en œuvre à chaque fois que l'on veut faire passer à 1 les chiffres 0 et à 0 les chiffres 1. Supposons que l'on ait un nombre dont on veuille faire changer d'état le dernier chiffre (bit 0) : on le combinera avec XOR 1. Si le nombre se terminait par 0, il se terminera alors par 1 mais par contre, si son dernier chiffre était 1, ce sera du coup 0. On essaie :

PRINT 28 XOR 1 ; réponse : **29**

```
     11100 <---------- 28
XOR 00001 <---------- 1
---------
=    11101 <---------- 29
```

PRINT 31 XOR 1 ; réponse : **30**

```
     11111 <---------- 31
XOR 00001 <---------- 1
---------
=    11110 <---------- 30
```

Bien entendu, XOR peut être utilisé pour faire basculer d'un état à l'autre n'importe lequel des chiffres sans modifier les autres. Par exemple, XOR 5 ne changera les états que du premier et du troisième chiffres en partant de la droite (5 est égal à 101 en binaire).

2

LA MÉMOIRE ÉCRAN

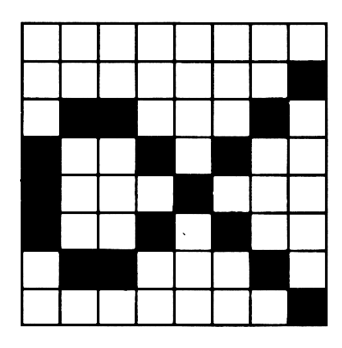
L'Amstrad transmet au téléviseur une image dont les caractéristiques sont données par l'instruction MODE. Trois types de résolution sont permis et à chacun d'eux est associé un nombre bien déterminé de couleurs. Ces modes ne peuvent cohabiter dans l'ordinateur car ils utilisent tous trois la même portion de mémoire vive pour garder la copie de l'image qui apparaît sur l'écran.

STRUCTURE DE LA MÉMOIRE ÉCRAN DE L'AMSTRAD

Sur la partie utilisable de l'écran on peut faire entrer au maximum 80 caractères en largeur et 25 en hauteur, ce qui fait que l'on peut écrire au total 80×25 soit 2 000 caractères.

Caractères définis par l'utilisateur

Prenons le temps de rappeler la configuration d'un caractère et indiquons, à titre d'exemple, la définition du symbole α.



Premier segment
Deuxième segment
Troisième segment

Huitième segment

Le caractère est inscrit dans un carré de 64 cases. On y a noirci celles qui feront ressortir la lettre α et on va demander à l'ordinateur de n'allumer que ces cases-là. Il serait naturellement fastidieux de prendre les cases les unes à la suite des autres et de préciser à chaque fois si elles doivent être allumées ou éteintes. Aussi la définition d'un caractère se fait-elle segment par segment, en ayant à l'esprit que la convention suivante a été décidée une fois pour toutes : cha-

que case allumée sera représentée par le chiffre 1 et chaque case éteinte par le chiffre 0.

Ainsi les huit cases de la première ligne seront notées 00000000, les huit cases de la deuxième ligne seront notées 00000001, celles de la troisième 01100010, etc. La forme finale de la définition de ce caractère sera donc :

```
&B00000000,&B00000001,&B01100010,&B10010100,
&B10001000,&B10010100,&B01100010,&B00000001
```

Le symbole &B placé devant chaque liste de 0 et de 1 indique qu'il s'agit d'un nombre binaire. L'étude du Chapitre 1 a fait comprendre au lecteur pour quelle raison nous sommes autorisés à écrire plus simplement :

```
SYMBOL 255,0,1,98,148,136,148,98,1
```

On retient de ceci que chaque caractère est constitué de huit segments superposés et on donne le résultat suivant : l'image fournie par l'ordinateur est formée de 16 000 segments. A chacun de ceux-ci correspond un octet de la mémoire centrale.

Il nous faut voir maintenant comment le programmeur va avoir la possibilité d'agir sur ces segments. Dans un premier temps, c'est l'instruction POKE qui sera employée puis, très vite, viendra le moment d'expérimenter vos connaissances de l'assembleur.

Correspondance entre un segment et son adresse en mémoire

Comme il vient d'être dit, à chacun des 16 000 segments est associé un octet c'est-à-dire une case mémoire pouvant prendre une valeur comprise entre 0 et 255. Ce dernier nombre se déduit directement du fait qu'un octet est un ensemble de huit bits et qu'un bit ne peut prendre que la valeur de l'un des deux chiffres binaires 0 et 1. 11111111 vaut bien 255 en décimal, n'est-ce pas ?

Pour agir sur un octet, il faut connaître l'adresse de l'octet correspondant et écrire dans cet octet, avec POKE, l'information voulue. Nous allons voir des exemples dans quelques instants mais il faut d'abord comprendre la relation qu'il y a entre l'emplacement d'un

segment sur l'écran et le numéro (ou adresse) de l'octet qui lui est associé en mémoire.

Le premier segment, celui qui est disposé tout à fait en haut et à gauche de l'écran, a pour adresse 49152. Cette valeur a été choisie par le constructeur de notre ordinateur pour des raisons que nous n'avons pas à connaître. Nous avons juste à savoir que si nous modifions le contenu de l'octet 49152, le dessin du petit segment situé en haut et à gauche s'en trouvera changé. Comment ? Patience...

Le deuxième segment, placé immédiatement à droite du premier, a pour adresse 49153.

Le troisième, à droite du deuxième, a pour adresse 49154.

Le quatre-vingtième a pour adresse 49231 (c'est-à-dire 49152 + 79). Il est dessiné à droite et tout en haut de l'écran.

Nous venons de nous occuper d'une ligne horizontale complète, la première. Reste à voir les 199 suivantes (199 = 25 × 8 – 1). La bonne logique voudrait que la deuxième ligne débute avec un segment ayant pour adresse 49232, non ? Hélas, ce serait trop simple ! C'est en réalité le premier segment de la neuvième ligne qui a pour adresse 49232. Puis, tout a l'air de rentrer dans l'ordre :

Le deuxième segment de la neuvième ligne a pour adresse 49233.

Le quatre-vingtième segment de la neuvième ligne a pour adresse 49311 (soit 49232 + 79).

Et, de nouveau, sept lignes complètes seront sautées :

Le premier segment de la dix-septième ligne a pour adresse 49312.

Le dernier segment de cette même ligne a pour adresse 49391, etc.

Tout ceci ne vous dit pas comment est obtenu l'octet relatif au premier segment de la deuxième ligne. Voici la réponse : on ajoute 2048 à l'adresse du segment placé juste au-dessus (donc à 49152) et l'on obtient le résultat recherché, 51200. Ne me demandez pas pourquoi on ajoute 2048 et pas un autre nombre ; je vous répondrais que cette valeur a été choisie par le constructeur de notre ordinateur pour des raisons...

Continuons notre travail de déchiffrage de la mémoire vidéo.

L'octet 51201 aura la garde du deuxième segment de la deuxième ligne.

L'octet 51279 surveillera le dernier segment de cette même ligne.

Quant au premier segment de la troisième ligne, on calculera son adresse en ajoutant... 2048 à l'adresse du segment immédiatement supérieur. Vous voyez qu'il y a tout de même une certaine logique dans la mémoire écran de l'Amstrad !

Voici le résumé de ce que l'on vient de voir :

49152	49153	. . . .	49231	1re ligne
51200	51201	. . . .	51279	2e ligne
53248	53249	. . . .	53327	3e ligne
.			.	4e ligne
.			.	5e ligne
.			.	6e ligne
.			.	7e ligne
.			.	8e ligne
49232	49233		49311	9e ligne
.			.	
.			.	
.			.	
49312	49313		49391	17e ligne
.			.	
.			.	
.			.	
.			.	
.			.	
65408	65409		65487	200e ligne

LA MÉMOIRE ÉCRAN EN MODE 2

Seules deux couleurs sont disponibles dans ce mode de haute résolution. Chaque caractère affiché a exactement la largeur d'un segment et chaque point d'un segment ne peut avoir que la couleur définie par la commande PAPER ou celle déterminée par PEN.

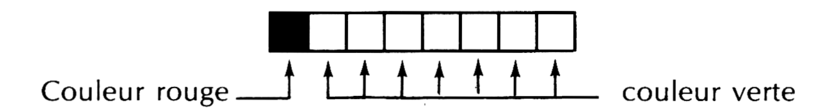
Faites entrer le programme suivant dans votre ordinateur et exécutez-le :

```
10 MODE 2 : INK 0,9 : INK 1,3 : BORDER 9
20 FOR I = 49152 TO 49191
30 POKE I , 128
40 NEXT : LOCATE 1,10
```

Vous devez voir en haut de l'écran devenu vert une succession de 40 points rouges. C'est l'instruction POKE qui a écrit dans les 40 premiers octets de la mémoire vidéo la valeur 128. Voyons à quoi cela correspond :

128 décimal = 10000000 binaire

Cette égalité nous permet d'en déduire que seul le point de gauche du premier segment est allumé en rouge.



Bien entendu, puisque nòtre programme boucle 40 fois, il est possible de donner une explication complète : chacun des 40 premiers octets de la mémoire écran a vu son bit de gauche mis à 1 et les autres abaissés à 0. Et les 40 premiers segments se sont retrouvés avec leur point de gauche allumé. Ne prêtez aucune attention particulière à la commande LOCATE de la ligne 40 ; elle n'est là que pour empêcher le message *Ready* d'apparaître au milieu des points que nous allumons.

Un deuxième exemple. Remplacez la ligne 30 par

```
30 POKE I, 240
```

L'exécution du programme doit faire apparaître devant vos yeux une ligne tracée en pointillé. La raison en est que 240 s'écrit 11110000 en binaire et que chacun des 40 premiers segments vidéo se voit pour moitié dessiné en rouge (portion de gauche) et pour moitié en vert (portion de droite).

Essayez d'écrire avec POKE d'autres valeurs dans l'octet I. Par exem-

ple 0 (aucun point ne s'éclaire) ou 255 (tous les points sont tracés en rouge et une ligne complète se dessine).

Afin de vous assurer que tout ceci est bien assimilé, essayez de prévoir ce qui va apparaître sur votre écran si vous faites exécuter les lignes suivantes :

```
10 MODE 2 : INK 0,9 : INK 1,3 : BORDER 9
20 FOR I = 49152 TO 49158
30 POKE I + J , 255 : J = J + 2048
40 NEXT : LOCATE 1, 10
```

Pour en terminer avec ce mode, essayons d'utiliser ce qui vient d'être établi pour dessiner la lettre α au milieu et en haut de l'écran. Voici ce que l'on avait obtenu comme ensemble de nombres pour définir ce caractère :

0 , 1 , 98 , 148 , 136 , 148 , 98 , 1

On choisit comme octet correspondant au segment du haut de ce symbole l'octet 49192. Il est situé sur la première ligne et à peu près au milieu de l'image. Il faut écrire dans cet octet la valeur 0 car la première ligne est constituée de cases vides.

```
POKE 49192 , 0
```

La ligne 2 dans laquelle il faudra allumer le point de droite correspond à l'octet 49192 + 2048.

```
POKE 51240 , 1
```

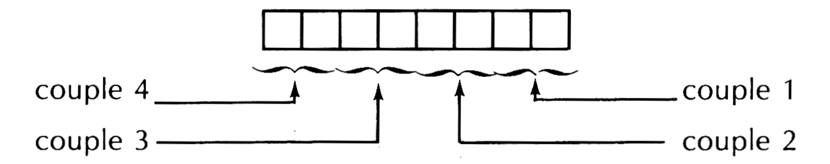
En augmentant les adresses des octets par pas de 2048 et en y inscrivant à chaque fois les valeurs convenables, on arrivera au dessin complet de la lettre α sur l'écran. Le programme qui suit retrace dans une boucle les huit opérations nécessaires.

```
10 MODE 2 : INK 0 , 9 : INK 1 , 3 : BORDER 9
20 FOR I = 49192 TO 63528 STEP 2048
30 READ J : POKE I , J : NEXT
40 DATA 0 , 1 , 98 , 148 , 136 , 148 , 98 , 1
```

Le nombre 63528 qui est écrit dans la ligne BASIC 20 est égal, mais vous l'aviez pressenti, à 49192 + 2048 * 7.

LA MÉMOIRE ÉCRAN EN MODE 1

Dans le mode précédent, il a été vu que les huit points de chaque segment pouvaient s'allumer ou s'éteindre indépendamment les uns des autres. Ce n'est plus le cas maintenant ; en mode 1 un segment doit être considéré comme la somme de quatre paires de points. Ce que nous perdons ainsi en finesse de résolution va être compensé par le fait que quatre couleurs sont mises à notre disposition.



Nous nous comprenons bien, n'est-ce pas ? Les deux points de droite (couple numéro 1) auront obligatoirement la même couleur. Et il en ira de même pour n'importe quelle paire de points appartenant au même couple. C'est ce qui nous permet de dire qu'un segment peut avoir quatre couleurs différentes (une pour chaque couple).

Bien, tapez maintenant le programme suivant :

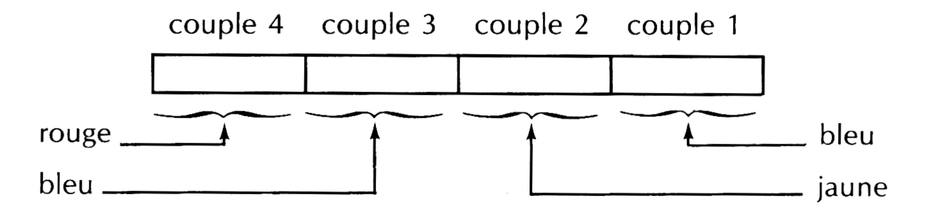
```
10 MODE 1 : INK 0,1 : INK 1,24
20 INK 2,3 : INK 3,9
30 FOR I = 53248 TO 53297
40 POKE I , 40
50 NEXT : LOCATE 1,10
```

Les quatre instructions INK du programme BASIC correspondent aux couleurs suivantes :

INK 0 : BLEU
INK 1 : JAUNE
INK 2 : ROUGE
INK 3 : VERT

La boucle FOR NEXT écrit dans chacun des octets ayant une adresse comprise entre 53248 et 53297 le même nombre décimal 40. Les 50 octets choisis sont en relation avec les 50 premiers segments de la troisième ligne du téléviseur.

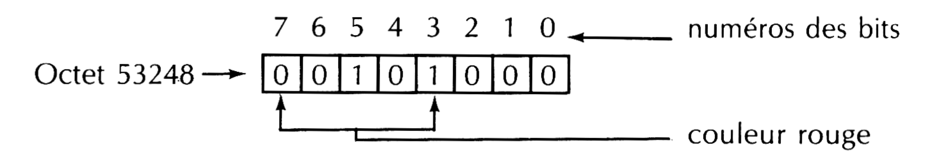
L'exécution du programme nous montre, si l'on a une bonne vue, que chaque segment est coloré de la façon suivante :



Vous voyez certainement le rapport qu'il y a entre cette coloration et le nombre 40 ! Non ? Essayez alors le binaire :

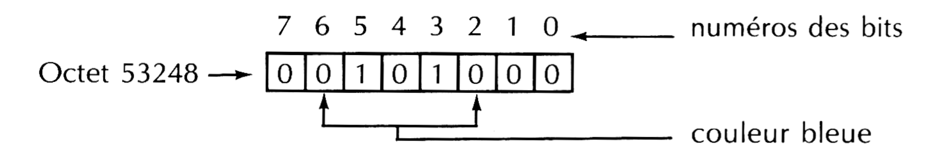
40 décimal = 00101000 binaire

Vous ne voyez toujours rien ? Décidément notre ordinateur a tout fait pour que l'on ne perce pas ses mystères, y compris d'inventer un codage des couleurs tellement compliqué qu'on ne risque pas, seul, d'en venir à bout. Cela dit, voici la réponse :



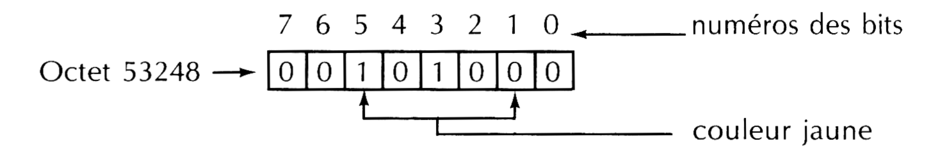
Les bits 3 et 7 de l'octet 53248 donnent la couleur du couple le plus à gauche, celui qui s'allume en rouge donc. Ces bits ont pour valeurs binaires 1 et 0 (respectez bien l'ordre : bit 3 puis bit 7) et l'on obtient donc le nombre binaire 10, c'est-à-dire 2 en décimal. Il ne reste plus alors qu'à remarquer que INK 2 donne justement la couleur rouge.

La couleur suivante maintenant :



La couleur du couple de points numéro 3 est donnée par les bits 2 et 6. On aura compris pourquoi cette couleur est le bleu quand on aura vu le rapport direct qui existe entre l'écriture binaire 00 (0 décimal donc) et la commande INK 0 (teinte bleue).

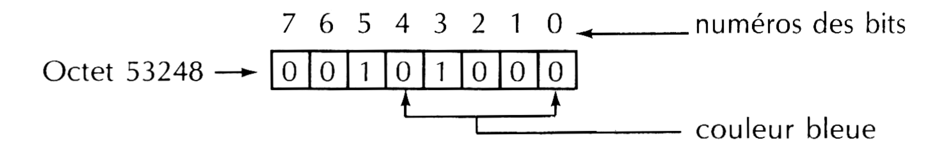
Continuons avec le couple 2 :



Le nombre binaire qui correspond à la couleur du couple 2 vaut 01. On l'obtient grâce aux bits 1 et 5.

Or 01 binaire est égal à 1 décimal ; cela, ajouté au fait que INK 1 correspond au jaune, nous permet de comprendre pourquoi le couple numéro 2 apparaît coloré en jaune.

Le dernier couple maintenant :



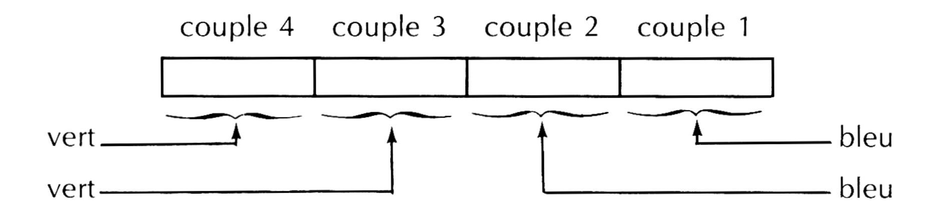
Ce sont cette fois les bits 0 et 4 qui donnent la couleur du couple de droite. Il apparaît sur l'écran avec la couleur bleue.

D'autres exemples :

Remplaçons la ligne 40 par

```
40 POKE I , 204
```

Voici comment chacun des 50 segments de la troisième ligne du téléviseur seront vus :



Cela s'explique par l'égalité suivante :

204 décimal = 11001100 binaire

	7	6	5	4	3	2	1	0	← numéros des bits
Octet 53248 →	1	1	0	0	1	1	0	0	

Couple 4 : les bits 3 et 7 donnent la valeur binaire 11, c'est-à-dire 3 décimal. Ce couple s'allume donc en vert.

Le couple 3 s'allume lui aussi en vert pour les mêmes raisons (bits 2 et 6).

Les couples 2 et 1 apparaissent en bleu car la valeur binaire qui leur est associée est égale à 00.

Un nouvel exemple :

```
40 POKE I , 240
```

Une seule ligne jaune apparaît sur l'écran et voici pour quelle raison :

240 décimal = 11110000 binaire

	7	6	5	4	3	2	1	0	← numéros des bits
Octet 53248 →	1	1	1	1	0	0	0	0	

Bits 3 et 7 : couleur jaune (valeur 01)
Bits 2 et 6 : idem
Bits 1 et 5 : idem
Bits 0 et 4 : idem

Un dernier exemple : conservez les lignes 10, 20 et 50 du programme précédent et modifiez les lignes 30 et 40 comme suit :

```
30 FOR I = 49152 TO 63488 STEP 2048
40 POKE I , 195 : POKE I + 1 , 255
```

Sachant que 195 et 255 s'écrivent respectivement 11000011 et 11111111 en binaire et que 63488 est égal à 49152 + 2048 * 7, pouvez-vous prévoir ce qui sera visible dans le coin supérieur gauche de votre écran ?

LA MÉMOIRE ÉCRAN EN MODE 0

Dans le mode de basse résolution, chaque segment de l'écran est divisé en deux. Les quatre points de gauche ont obligatoirement la même couleur et cela est vrai aussi pour les quatre points de droite. On voit donc qu'un segment ne peut prendre que deux couleurs différentes mais, en contrepartie, ces deux couleurs peuvent être choisies parmi une palette de 16 teintes.

Voici le premier exemple :

```
10 MODE 0 : INK 0,1 : INK 1,15
20 INK 7,3 : INK 12,9 : INK 15,0
30 FOR I = 53248 TO 53297
40 POKE I , 213
50 NEXT : LOCATE 1,10
```

Les cinq couleurs que nous nous proposons d'utiliser dans ce paragraphe sont :

INK 0 : BLEU
INK 1 : ORANGE
INK 7 : ROUGE
INK 12 : VERT
INK 15 : NOIR.

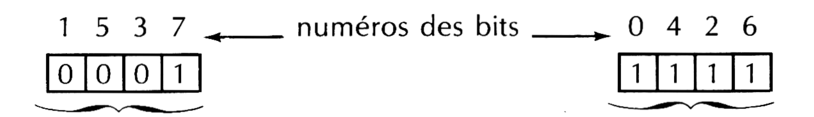
Pour l'instant, seules trois d'entre elles sont visibles sur le téléviseur. Le bleu est la couleur du fond et les petits segments qui se succèdent sur la troisième ligne de l'écran sont colorés pour moitié en orange et pour moitié en noir. La façon dont l'ordinateur a interprété la valeur décimale 213 pour déterminer les couleurs associées aux segments n'est pas, vous vous en doutez, tout à fait évidente.

213 décimal = 11010101 binaire

L'octet 53248 et ses 49 suivants ont donc la structure :

	7	6	5	4	3	2	1	0 ← numéros des bits
Octet 53248 →	1	1	0	1	0	1	0	1

Vous seriez capables de retrouver les couleurs orange (INK 1) et noire (INK 15) dans cet ensemble de chiffres 0 et 1 ? Non, naturellement, car l'Amstrad a une façon bien à lui de combiner les bits. Voyez plutôt :



couleur orange — couleur noire

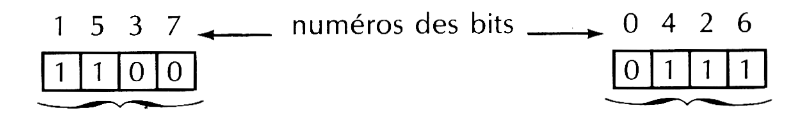
Remettez tout cela dans l'ordre et vous retrouverez la configuration de l'octet 53248 : bit 7 à 1, bit 6 à 1, bit 5 à 0, etc. D'autre part, le nombre binaire obtenu avec les bits 1, 5, 3 et 7 vaut 0001 soit 1 en décimal. C'est bien la couleur orange que nous venons de retrouver ainsi. Pour leur part, les bits 0, 4, 2 et 6 font apparaître le nombre 1111. Sa traduction décimale vaut 15 et cela explique pourquoi la partie droite de chaque segment est tracée en noir.

Plutôt compliqué, comme méthode. La vie des programmeurs aurait été rendue plus facile si par exemple les bits 7, 6, 5 et 4 avaient servi à déterminer la couleur de la partie gauche et les bits 3, 2, 1 et 0 celle de la partie droite. Enfin, on finit par s'y habituer !

Un deuxième exemple. Décidons que cette fois les segments devront apparaître en vert et en rouge, c'est-à-dire en définitive avec les encres 12 et 7 :

12 décimal = 1100 binaire (vert)
 7 décimal = 0111 binaire (rouge)

Il ne reste qu'à placer ces décompositions binaires dans les bits voulus :



couleur verte — couleur rouge

On réécrit ensuite les bits dans l'ordre normal :

	7	6	5	4	3	2	1	0 ← numéros des bits
Octet 53248 →	0	1	1	1	0	1	1	0

et l'on traduit ce résultat en décimal :

01110110 binaire = 118 décimal

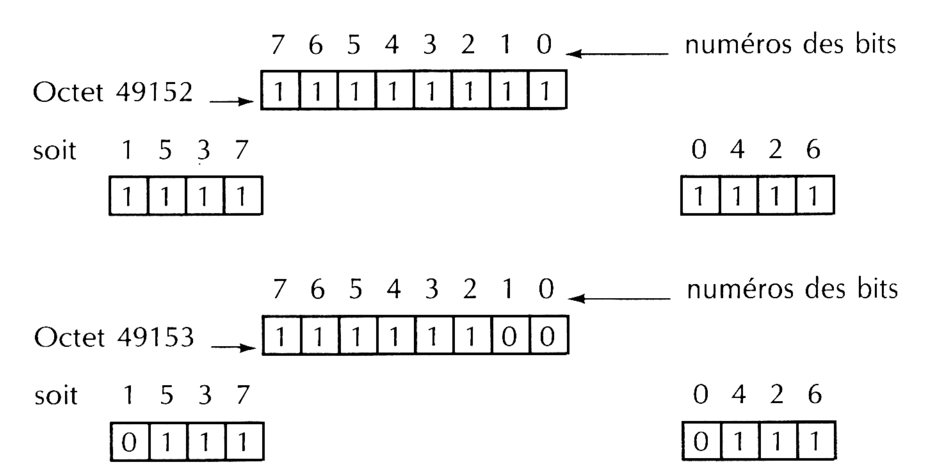
Conclusion : la ligne BASIC 40 devra être remplacée par :

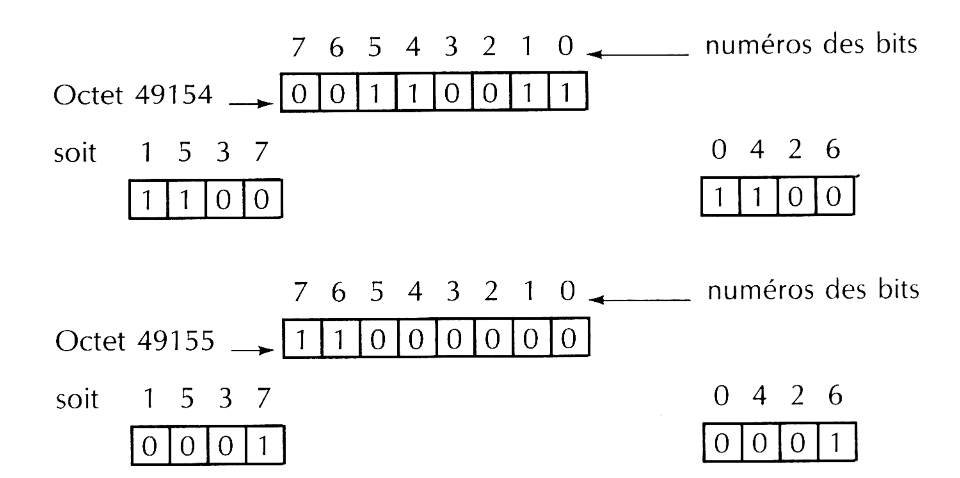
```
40 POKE I , 118
```

Le dernier exemple maintenant. Étudiez le programme suivant et essayez de comprendre pourquoi son exécution va avoir pour effet d'allumer les quatre premières cases caractères de l'écran avec des couleurs différentes.

```
10 MODE 0 : INK 0,1 : INK 1,15
20 INK 7,3 : INK 12,9 : INK 15,0
30 FOR I = 49152 TO 63488 STEP 2048
40 POKE I,255 : POKE I+1,252
50 POKE I+2,51 : POKE I+3,192
60 NEXT : LOCATE 1,10
```

Aidez-vous des indications suivantes :





ET POUR ALLER PLUS LOIN

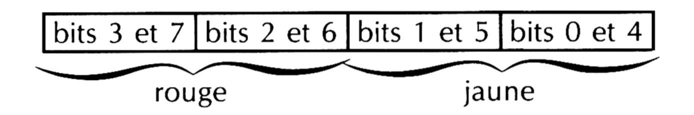
La description qui vient d'être donnée concernant la répartition des octets de la mémoire écran, pour compliquée qu'elle soit, n'en est pas moins incomplète. Nous avons toujours considéré que l'octet 49152 correspondait au premier segment de l'image, segment situé en haut et à gauche. Vous pensez que l'Amstrad, le connaissant comme nous le connaissons, va s'en tenir là ? Non, bien sûr ; pour lui l'octet 49152 n'a pas à s'occuper spécialement du premier segment vidéo. Vous pourrez, plus tard, vous reporter à la documentation technique jointe à votre machine. Pour l'instant, la seule chose que nous ayons à savoir est que, lorsque l'ordinateur rencontre la commande MODE, il met automatiquement en relation l'octet 49152 et le premier segment de l'écran. Aussi prendrons-nous l'habitude d'inclure une instruction MODE dans la première ligne de nos programmes à chaque fois qu'un effet graphique sera recherché.

PROGRAMME DE DÉMONSTRATION

Pour clore ce chapitre, il est proposé une comparaison entre les vitesses d'exécution du même programme, suivant que celui-ci a été

écrit en BASIC ou en assembleur. Gageons que la comparaison ne tournera pas à l'avantage de la version BASIC.

Le but des lignes qui suivent est d'allumer chacun des 16 000 segments de l'écran. Le programme fonctionnant en mode 1 et 48 s'écrivant 00110000 en binaire, on en déduit que chacun de ces segments sera bicolore.




```
 10 MODE 1 : INK 0,3 : INK 1,24
 20 FOR I = 49152 TO 51151
 30 FOR J = 0 TO 7 : K = J*2048
 40 POKE I + K , 48 : NEXT : NEXT
 50 CLS : LOCATE 1,10
 60 PRINT " VOICI LA VERSION ASSEMBLEUR "
 70 FOR I = 1 TO 2000 : NEXT : MEMORY 43800
 80 FOR I = 43801 TO 43827 : READ A$
 90 POKE I , VAL("&H" + A$) : NEXT
100 CLS : CALL 43801
110 DATA 11,00,08,21,00,C0,E5,06,08,36,30,19,10,FB
120 DATA E1,23,7C,FE,C7,20,F1,7D,FE,D0,20,EC,C9
```

La partie qui correspond à la version BASIC se comprend aisément et est la mise en application de ce qui a été vu dans les paragraphes précédents. Puisqu'on écrit par POKE le nombre 48 dans chacun des octets vidéo, l'écran va se recouvrir de segments rouge et jaune. La superposition puis la juxtaposition de ces segments fera apparaître sur l'image, une fois le programme exécuté, une succession de lignes verticales bicolores.

Reste à voir dans quel ordre les segments s'allument.

Lorsque I vaut 49152, le premier segment se colorie. Les 7 segments qui se trouvent au-dessous de lui, et qui sont obtenus en respectant à chaque fois un écart de 2 048 unités, subissent le même traitement.

C'est ensuite au tour du deuxième segment de l'écran (octet 49153) de se rendre visible. Il est aussitôt imité par les 7 segments placés immédiatement au-dessous.

Et ce processus se renouvelle jusqu'à ce que la variable I prenne la valeur 49231. Les 8 premières lignes horizontales sont alors complètement dessinées sur le téléviseur.

A ce moment-là, le programme retrouve le premier segment de la neuvième ligne (octet 49232). Il l'allume, ainsi que les 7 segments suivants et passe à l'octet 49233.

Est-il besoin d'en dire plus ? Simplement, pour vous éviter de chercher comment s'obtient la dernière valeur que prend la variable de boucle I, nous donnons la réponse :

$$51151 = 49152 + 80 * 25 - 1$$

(80 est le nombre de segments par ligne et 25 est le nombre de caractères qui entrent en hauteur dans l'écran.)

Les explications permettant de comprendre la partie langage machine de ce programme sont données maintenant. Inutile de vous y intéresser, vous risqueriez de vous décourager devant ce que vous auriez tendance à considérer comme étant d'une difficulté insurmontable. Passez donc au chapitre suivant et soyez assurés que, dans quelques jours, ce qui suit vous paraîtra limpide.

Nombre d'octets : 27.

Lignes	Codes machine	Assembleur		
1	11 00 08		LD	DE,2048
2	21 00 C0		LD	HL,49152
3	E5	DEBUT:	PUSH	HL
4	06 08		LD	B,8
5	36 30	SUITE:	LD	(HL),48
6	19		ADD	HL,DE
7	10 FB		DJNZ	SUITE (–5)
8	E1		POP	HL
9	23		INC	HL
10	7C		LD	A,H
11	FE C7		CP	199
12	20 F1		JR	NZ,DEBUT (–15)
13	7D		LD	A,L
14	FE D0		CP	208
15	20 EC		JR	NZ,DEBUT (–20)
16	C9		RET	

Lignes 1 et 2 : les registres DE et HL sont initialisés. Le premier gardera tout au long du programme la même valeur 2048. C'est, rappelons-le, l'écart qui sépare deux segments placés l'un en dessous de l'autre. Le second registre, HL, va contenir successivement les adresses de tous les octets de la mémoire écran.

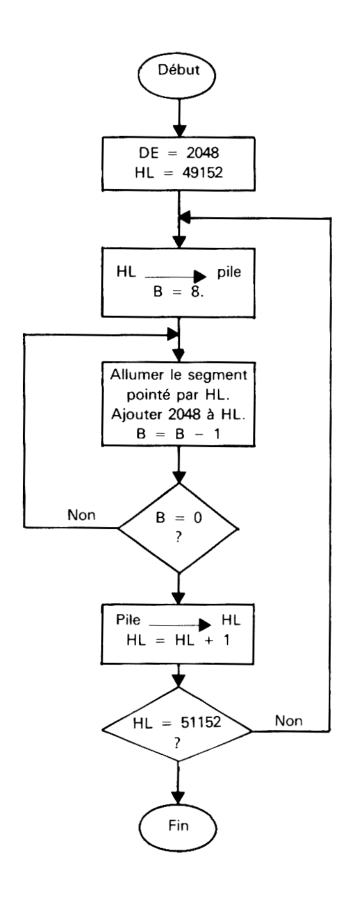
Lignes 3 et 4 : on sauvegarde dans la pile la valeur 49152 et on charge le registre B avec le nombre 8.

Lignes 5, 6 et 7 : 48 est écrit dans l'octet 49152, ce qui a pour effet d'allumer en rouge et jaune le segment correspondant. On recommence la même opération avec chacun des 7 segments dont les adresses s'échelonnent de 2048 en 2048. Lorsque cela est fait, la première case caractère de l'image est entièrement colorée.

Lignes 8 et 9 : on retrouve la valeur initiale de HL, soit 49152, et on lui ajoute une unité. L'ordinateur, retournant à la ligne 3, va donc allumer les 8 segments de la deuxième case caractère. La logique du programme assembleur est, comme on le voit, rigoureusement identique à celle du programme BASIC : on colorie chacun des segments de l'intervalle 49152-51151 et on en profite à chaque fois pour allumer les 7 segments placés immédiatement en dessous.

Lignes 10 à 16 : le programme doit boucler jusqu'à ce que le registre HL arrive à 51152. Le poids fort de cette valeur est égal à 199 et il faut donc rebrancher le microprocesseur à la ligne DEBUT tant qu'elle n'est pas atteinte. Dès que c'est le cas, on attend que le poids faible du registre HL vale 208 pour sortir de la routine et laisser le BASIC reprendre son cours.

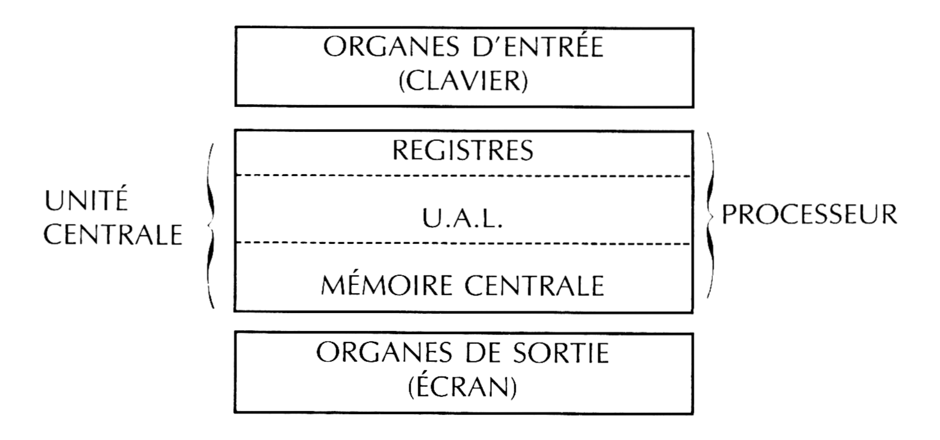
Voici l'organigramme relatif à ce programme :



3

L'ARCHITECTURE INTERNE DU MICROPROCESSEUR Z80

Avant de commencer l'étude du microprocesseur, il est nécessaire de le situer dans son environnement : aussi ce chapitre débutera par quelques rappels sur la structure générale d'un micro-ordinateur.



C'est essentiellement la mémoire centrale qui sera utile à notre étude et c'est pour cette raison que les mémoires auxiliaires n'apparaissent pas sur le schéma.

LA MÉMOIRE CENTRALE

La mémoire centrale permet d'enregistrer, de conserver et de restituer à la demande les informations qui lui ont été communiquées. Ces informations sont de deux sortes : ce sont soit des données soit des programmes. A priori, rien ne distingue en mémoire ces deux types d'informations et c'est la seule logique du programme qui empêchera la confusion.

Pour des raisons technologiques, l'information rangée en mémoire se trouve sous la forme de combinaisons des chiffres binaires 0 et 1. Le bit (ou chiffre binaire) est l'unité élémentaire d'information et ne peut prendre que l'une de ces valeurs. Un octet est constitué de huit bits. Le nombre le plus grand que l'on puisse avoir dans un octet

est le nombre 11111111 (soit 255 en décimal). Le nombre le plus petit est obtenu quand les huit bits valent 0 : c'est le nombre 00000000 (ou 0 en décimal).

La mémoire centrale de la plupart des micro-ordinateurs est divisée en 65536 octets et l'ordinateur est capable de retrouver le contenu de n'importe quel octet grâce à son adresse. Les octets de la mémoire sont numérotés de 0 à 65535 et c'est ce numéro que l'on appelle l'adresse. Il est possible de savoir le nombre que contient un octet avec la fonction BASIC PEEK. Essayons :

PRINT PEEK (45000) ; réponse : 126

Puisque 126 s'écrit 01111110 en binaire, on peut retrouver la configuration exacte de l'octet numéro 45000 :

0	1	1	1	1	1	1	0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

On peut dire aussi que cet octet contient le nombre hexadécimal 7E.

Puisqu'on est capable de connaître la valeur qui se trouve dans un octet, on doit être capable de modifier cette valeur : l'instruction POKE est à notre disposition pour cela.

PRINT PEEK (30000) ; réponse : 0
POKE 30000, 100
PRINT PEEK (30000) ; réponse : 100

Avant notre intervention, l'octet numéro 30000 contenait le nombre 0. Tous les octets contiennent une valeur et il est difficile de dire, dans l'absolu, à quoi elle correspond. Nous avons inscrit par POKE la valeur 100 dans l'octet et il ne nous est plus resté qu'à en demander la confirmation avec PRINT PEEK (30000).

Faisons un nouvel essai en faisant attention de ne pas chercher à inscrire dans un octet un nombre supérieur à 255 ; cela aurait pour conséquence de faire afficher le message *improper argument*. La raison en est qu'avec huit bits on ne peut constituer un nombre supérieur à 11111111 soit 255 en décimal.

Revenons à notre essai :

```
PRINT PEEK (49150) ; réponse : 137
POKE 49150 , 100
```

L'ordinateur ne nous laisse pas le loisir de taper :

```
PRINT PEEK (49150)
```

Il s'est bloqué, les touches ne répondent plus. Il se peut même que des dessins sans signification apparaissent sur l'écran. L'opération que nous venons d'accomplir s'appelle ordinairement un *plantage.* Cela ne risque en aucune façon d'abîmer notre machine mais, en tout cas, la seule chose que l'on puisse faire d'elle désormais est de l'éteindre. Elle aura repris tous ses esprits quand, dans une dizaine de secondes, nous la rallumerons.

Que faut-il retenir de cette malheureuse expérience ? Eh bien que certains octets de la mémoire sont utilisés de façon interne par l'ordinateur et qu'il vaut mieux, dans un premier temps du moins, ne pas chercher à les modifier.

Voici, pour les esprits curieux, une première approche de la carte de la mémoire vive de l'Amstrad. Les portions notées *Système* n'ont pas normalement à être modifiées par le programmeur. La partie supérieure est déjà connue de nous : c'est la zone de la mémoire écran. La partie placée entre les octets 367 et 43903 est, quant à elle, réservée au stockage du programme BASIC de l'utilisateur.

Mémoire vidéo	65535 49152
Système	
Mémoire utilisateur	43903 367
Système	0

L'UNITÉ ARITHMÉTIQUE ET LOGIQUE (U.A.L.)

L'UAL est constituée de circuits électroniques câblés et est capable d'effectuer des calculs arithmétiques (additions et soustractions) et des choix logiques (comparaison de deux nombres). Elle permet aussi de faire des opérations de décalage et de rotation, opérations auxquelles quelques pages de ce livre seront consacrées plus loin.

En écrivant nos programmes, nous n'aurons pas à intervenir directement sur l'UAL mais, même s'il n'en est plus du tout question explicitement, il ne faudra pas oublier le rôle capital de cette unité dans un ordinateur : l'UAL est le calculateur du microprocesseur.

LES REGISTRES

A 8 bits	
B 8 bits	C 8 bits
D 8 bits	E 8 bits
H 8 bits	L 8 bits
IX 16 bits	
IY 16 bits	
SP 16 bits	
PC 16 bits	

Les registres sont identifiables à des cases mémoire par lesquelles l'information va transiter. Le microprocesseur Z80 contient des registres 8 bits et des registres 16 bits. Les instructions BASIC ne permettent pas l'accès à ces registres et nous ne pourrons jamais, avec la fonction PEEK par exemple, savoir ce que contient tel ou tel registre.

Les registres A, B, C, D, E, H et L

Ils sont tous constitués de 8 bits, ce qui fait que le plus grand nombre qu'ils peuvent contenir est 255 décimal (FF hexa ou 11111111 binaire).

a. Le registre A

C'est le plus utilisé en assembleur et on lui donne souvent le nom d'accumulateur. Nous seront obligés de passer par lui dès qu'il s'agira par exemple de faire un calcul ou d'effectuer une comparaison entre deux valeurs.

b. Les autres registres 8 bits

Il n'y aurait rien de spécial à dire sur ces registres s'ils n'avaient la particularité très importante de pouvoir être associés par deux pour constituer des registres doubles.

Voyons cela de plus près. Supposons que les bits des registres B et C soient disposés comme suit :

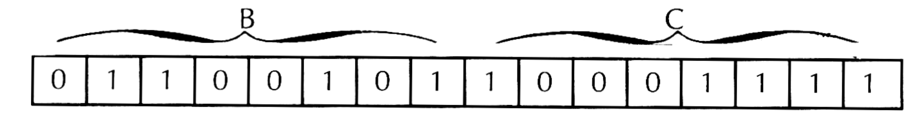
	128	64	32	16	8	4	2	1
B	0	1	1	0	0	1	0	1

Dans B se trouve donc le nombre décimal 101 ou hexadécimal 65.

	128	64	32	16	8	4	2	1
C	1	0	0	0	1	1	1	1

Dans C, c'est le nombre 143 (décimal) ou 8F (hexa) qui se trouve.

Pour former le registre BC, on associe B et C :



La valeur de BC est alors 0110010110001111, c'est-à-dire 25999 en décimal. Il faudra toujours se souvenir que BC est un registre 16 bits

et qu'il correspond donc à deux octets (soit un nombre compris entre 0 et 65535).

L'exemple qui vient d'être donné aurait tout aussi bien pu se concevoir en remplaçant les registres B et C par D et E ou par H et L. Nous aurions alors obtenu les registres doubles DE ou HL.

Les registres IX et IY

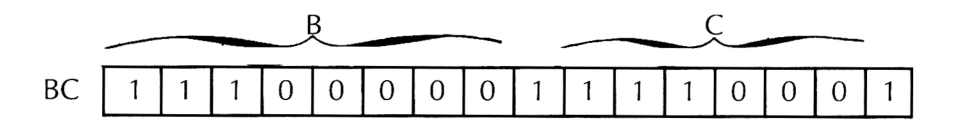
Ce sont des registres 16 bits et donc on peut y écrire n'importe quel nombre de 0 à 1111111111111111 (16 fois le chiffre 1). Si nous prenons la peine de traduire cette valeur binaire en décimal, on obtient 65535. Cela nous permet de comprendre le rôle que joueront IX et IY : ils contiendront généralement une adresse mémoire et cette adresse pourra être celle de n'importe quel octet de l'intervalle 0–65535.

IX et IY sont souvent appelés registres d'adresses, eu égard à ce que nous venons de dire, mais on les appelle aussi registres d'index car on peut les utiliser dans le mode d'adressage indexé (nous verrons cela bientôt).

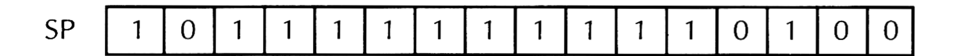
Le registre SP

Il porte le nom de pointeur de pile. Une pile est un endroit de la mémoire où seront stockés — empilés — des nombres les uns à la suite des autres. La structure de la pile d'un ordinateur correspond tout à fait à celle d'une pile d'assiettes : on peut toujours rajouter une assiette sur la pile mais, si l'on veut en reprendre une, ce sera toujours la dernière posée que l'on devra récupérer (dernier entré, premier sorti).

L'utilisation de la pile n'étant pas évidente pour le programmeur qui fait ses premiers pas en assembleur, passons un peu de temps sur un exemple. Supposons que, dans le processeur, les registres BC et SP aient les valeurs suivantes :



Les registres B et C contiennent respectivement les nombres décimaux 224 (ou E0 hexa) et 241 (ou F1 hexa). De ce fait, le registre BC contient la valeur 57585 (E0F1 hexadécimal).



Dans SP se trouve le nombre décimal 49140 (BFF4 hexadécimal).

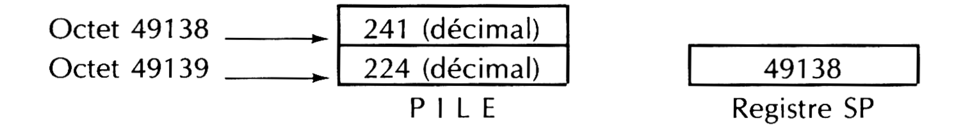
La valeur 49140 indique dans cet exemple que nous allons empiler nos nombres dans une série d'octets de la mémoire à partir justement de l'octet 49140. Cet octet en lui-même ne sera pas concerné par notre travail car c'est dans l'octet situé juste à côté que va démarrer notre empilement.

PUSH BC

Nous voilà devant notre première instruction assembleur. Elle utilise le mot anglais PUSH (pousser). Les lettres B et C précisent que c'est le contenu du registre BC qui va être placé sur la pile. Or, ne perdons pas de vue que la mémoire est constituée d'une succession d'octets (8 bits) et qu'il ne peut être question de placer le contenu du registre BC (16 bits) sur un seul octet. Qu'à cela ne tienne, la valeur de notre registre sera rangée en mémoire sur deux octets consécutifs, comme nous allons le voir.

Lorsque l'instruction PUSH BC aura été exécutée par le microprocesseur, voici ce qui se sera passé :

- Le registre BC n'aura subi aucune modification. Sa valeur, le nombre 57585, sera allée s'écrire dans la pile mais ce registre en aura conservé la trace. Il en sera toujours ainsi quand nous donnerons l'ordre au processeur de transférer un nombre d'un registre vers la mémoire : le registre ne sera pas modifié et tout se passera comme si le registre n'avait envoyé qu'un double, un duplicata à la mémoire.
- La pile sera formée de deux octets dont les contenus seront directement déduits de la valeur de BC. Les huit bits de gauche de ce registre (appelés bits de poids fort) seront recopiés dans l'octet 49139 et les huit autres bits (dits bits de poids faible) seront inscrits dans l'octet 49138. Ainsi tout se sera passé comme si l'on avait empilé d'abord le registre B et ensuite le registre C.

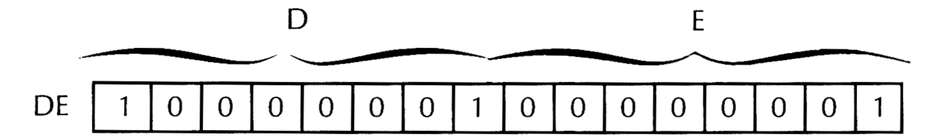


Comme nous le voyons, le registre SP vaut maintenant 49138. Voici donc défini son rôle : il contient l'adresse du dernier octet dans lequel une valeur a été empilée. On dit qu'il pointe sur le sommet de la pile.

Continuons avec

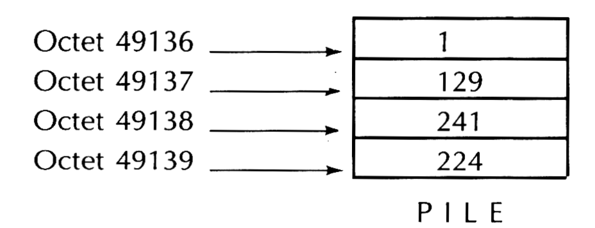
PUSH DE

Cette fois, c'est le contenu du registre DE qui va aller se placer sur la pile — c'est-à-dire dans les octets 49137 et 49136. Imaginons, pour fixer les idées, que DE ait la configuration suivante :



Le registre de poids fort D vaut alors 129 (décimal) et celui de poids faible 1. Le contenu de DE est donc égal à 33025.

Après exécution de l'instruction d'empilement par l'ordinateur, voici comment est constituée la pile :



Le registre SP va donc pointer sur l'octet 49136.

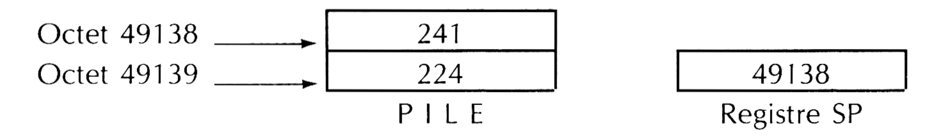
SP | 49136 |

Avant de passer à l'opération de dépilement, notons bien que les nombres sont stockés dans la pile sur des octets dont les adresses

décroissent à chaque fois. Il n'y a rien à y faire, c'est dans la logique sans doute de notre ordinateur : lui, il empile par en dessous. Le tout est de le savoir.

POP DE

C'est notre deuxième instruction assembleur : elle effectue exactement le travail inverse de la première. Elle va rechercher un nombre dans la pile et elle l'écrit dans le registre DE. Elle a donc dépilé les octets 49136 et 49137 et a été les replacer dans DE. Par là même, notre pile ne contient plus que deux octets et le registre SP repasse à 49138.



L'intérêt des instructions PUSH et POP n'apparaît pas immédiatement aux programmeurs débutant l'étude de l'assembleur car ils ne voient pas à quoi peuvent bien servir deux choses qui ne font rien d'autre que nous ramener à notre point de départ. Et pourtant c'est là justement qu'en réside tout l'intérêt : la principale difficulté de l'assembleur vient de ce que nous ne disposons que d'un nombre très réduit de registres. Nous verrons assez vite le très gros avantage qu'il y a à placer la valeur d'un registre dans la pile (PUSH), à utiliser ce registre pour faire autre chose, et à retrouver (POP) la valeur initiale de ce même registre.

Finissons-en avec notre exemple :

POP DE

La machine va aller écrire dans les registres E et D les nombres 241 et 224. Elle exécute très exactement l'ordre qu'on lui donne en prenant les nombres écrits au sommet de la pile — donc 241 et 224 — et en les plaçant dans E et D. Elle ne se soucie pas de savoir de quel registre proviennent ces deux valeurs. A ce moment-là, les deux registres BC et DE contiennent précisément la même valeur : 57585. La pile que nous avions constituée est réduite à zéro et le registre SP reprend sa valeur initiale 49140.

En fin de compte, dans notre exercice, le registre BC n'aura vu à aucun moment sa valeur modifiée mais une copie de son contenu aura été transférée dans le registre DE.

Un autre exemple maintenant :

Supposons que les doubles registres BC, DE et HL contiennent les valeurs décimales 1000, 2000 et 3000.

Quelles seront les valeurs de chacun d'eux quand la série d'instructions suivante aura été exécutée ?

```
PUSH  BC — PUSH  DE — PUSH  HL
POP   BC — POP   HL — POP   DE
```

Réponse : BC = 3000 ; HL = 2000 ; DE = 1000

Il faudra, dans les programmes assembleur, toujours avoir à l'esprit que la pile est utilisée de façon interne par le microprocesseur. Les ennuis nous seront garantis si, par mégarde, nous ne remettons pas la pile dans l'état où nous l'avons trouvée. Si on a eu besoin par exemple d'empiler six octets dans la pile, il faudra être sûr qu'à la fin de notre programme les six octets en question ont été dépilés.

Le registre PC

C'est un registre 16 bits que l'on appelle le compteur de programme ou le compteur ordinal. Le nombre qui est écrit dans ce registre est l'adresse de la prochaine instruction à exécuter. Il permet au microprocesseur de toujours savoir à quel octet du programme il va devoir s'intéresser. On ne l'utilise en programmation que dans des cas bien particuliers : il n'est pas directement accessible.

LES MODES D'ADRESSAGE

Un mode d'adressage est un moyen qui permet au microprocesseur d'avoir accès à une donnée. Cette donnée peut être un nombre quelconque dont on aura besoin dans le programme, un nombre qui se trouve déjà dans un registre, ou encore un nombre qui se trouve écrit quelque part en mémoire.

La connaissance des principaux modes d'adressage est obligatoire : elle permet d'écrire les programmes de la façon la plus courte, la plus simple et la plus lisible possible.

L'adressage inhérent

L'adressage inhérent est habituellement réservé aux instructions qui agissent directement sur les valeurs contenues par les registres. Ces instructions se comprennent d'elles-mêmes et n'ont aucunement besoin qu'on leur ajoute des indications.

Exemple : CPL.

Tous les microprocesseurs comprennent ce genre d'instruction : elle signifie que le registre A (lui et pas un autre) se verra complémenté. En clair, cela veut dire que chacun de ses bits prendra l'autre valeur binaire ; les chiffres 1 seront remplacés par des 0 et réciproquement.

A contenait 143 décimal (par exemple) ; soit 10001111 binaire.

```
CPL
```

A contient 112 décimal, c'est-à-dire 01110000 binaire.

L'adressage immédiat

Dans ce mode, une valeur apparaît après l'instruction assembleur. Prenons par exemple : LD A,100

La formule LD, qui sera retrouvée tout au long de ce livre, signifie que l'on va placer (charger) un nombre dans le registre A. Il est facile de voir qu'ici l'instruction LD n'aurait pas pu être écrite toute seule, comme dans l'adressage inhérent. Il nous faut absolument rajouter des indications à la suite : et, si l'on doit mettre un nombre dans l'accumulateur A, il faut bien dire lequel.

Dans le mode d'adressage immédiat, c'est la valeur marquée après la virgule (ici 100) qui sera écrite dans le registre que l'on aura choisi en écrivant son nom devant la virgule. Dans notre exemple, c'est le registre A qui est concerné mais n'importe quel autre registre 8 bits ou 16 bits aurait fait l'affaire.

A contenait, par exemple, 50

```
LD   A,100
```

A contient alors 100.
Un contre-exemple :

```
LD   A,300
```

Cette ligne devrait, en principe, écrire dans A le nombre 300. Vous l'aviez deviné ; cela n'a aucun sens puisque A est un registre 8 bits et que le nombre maximal qu'il peut contenir est 255.

L'adressage étendu

On l'appelle souvent adressage absolu car c'est un mode qui permet d'avoir accès directement au contenu de n'importe quel octet de la mémoire.

```
LD   A,(10)
```

L'accumulateur sera chargé non par le nombre 10, comme il l'aurait été dans l'adressage immédiat, mais avec la valeur écrite dans l'octet numéro 10.

```
PRINT PEEK(10) ; réponse : 185
```

En fin de compte, le registre A contiendra 185. Les parenthèses qui englobent la valeur 10 sont là pour éviter la confusion entre les modes immédiat et étendu. Nous respecterons ce type d'écriture dans tout le livre et elle aura toujours la même signification : lorsqu'une valeur sera «parenthésée», elle correspondra à l'adresse d'un octet de la mémoire et c'est le contenu de cet octet qu'il faudra prendre en considération.
Un deuxième exemple :

```
LD   A,(5000)
```

Un piège, comme dans le paragraphe précédent ? 5000 paraît bien trop important pour notre accumulateur 8 bits. Mais non, cette ligne n'est pas un non-sens : elle signifie que le registre A va contenir non

pas le nombre 5000 mais le nombre qui se trouve écrit dans l'octet ayant 5000 pour adresse.

```
PRINT PEEK(5000) ; réponse : 0
```

ce qui fait que A sera chargé, une fois l'instruction assembleur exécutée, par la valeur 0.

L'adressage registre

Ce mode d'adressage ne fait apparaître aucun nombre dans l'instruction et n'est utilisable que lorsque l'on veut intervenir sur les contenus des registres. Exemple :

```
INC   B
```

Cette instruction donne à l'ordinateur l'ordre d'incrémenter le registre B, c'est-à-dire d'augmenter sa valeur d'une unité.

B contenait 200 (par exemple)

```
INC B
```

B contient 201.

Un autre exemple :

```
LD   C,D
```

Nous retrouvons le mnémonique de chargement LD. Il n'y a aucune difficulté à interpréter cette instruction. Si 20 et 30 sont, par exemple, écrits dans les registres C et D, l'exécution de l'instruction inscrira la valeur 30 dans C (et D restera inchangé).

L'adressage indirect

Il est noté comme l'adressage étendu, mais l'expression qui se trouve entre parenthèses est, cette fois, le nom d'un registre.

Voici quelques exemples :

```
LD   C,(HL)
```

Le simple registre C va être chargé avec la valeur qui se trouve dans l'octet ayant pour adresse le nombre écrit dans HL.
Supposons que, dans ce registre 16 bits, il y ait le nombre 48000.

PRINT PEEK (48000) ; réponse : 146

Après LD C,(HL), il y aura dans le registre C la valeur 146. Cette instruction est donc tout à fait équivalente à LD C,(48000) (adressage étendu).
Deuxième exemple :

```
LD   HL,45000
LD   B,(HL)
```

Il faut considérer que la première instruction LD place directement dans le registre HL le nombre 45000 (mode d'adressage immédiat). La seconde instruction va recopier dans le registre B le contenu de l'octet 45000.

PRINT PEEK (45000) ; réponse : 222

B contiendra donc la valeur 222.
Faisons un nouvel essai :

```
LD   C,(IX+10)
```

Cette fois, le processeur va aller chercher, pour l'écrire dans C, la valeur qui se trouve en mémoire dans l'octet ayant pour adresse le nombre contenu dans IX auquel on ajoute 10. C'est plus facile à comprendre qu'à expliquer !
Admettons que IX contienne la valeur 46000. On ajoute 10 à 46000 et l'octet concerné est alors le numéro 46010.

PRINT PEEK (46010) ; réponse : 63

Ainsi, dans cet exemple, C va être chargé avec le nombre 63.
Un dernier exemple :

```
LD   H,(IY-20)
```

En considérant que IY contient le nombre 47000 (par exemple), cette instruction indique à la machine qu'elle doit placer dans H le nombre écrit dans l'octet 46080.

Si l'on sait que le PEEK de cet octet vaut 132, on en déduit que le registre H aura vu son contenu porté à 132.

Remarquons que seuls les registres IX et IY ont le privilège de pouvoir accéder à une case mémoire dont l'adresse est donnée par leurs contenus auxquels est ajoutée ou retranchée une certaine quantité. Les instructions du type :

```
LD   A,(BC+5)
```

ou

```
LD   A,(DE-3)
```

n'existent pas. C'est la raison pour laquelle on considère parfois que l'adressage indirect, lorsqu'il est relatif aux registres IX et IY, doit avoir un nom différent. On parle alors d'adressage indexé.

Nous en avons terminé avec ce chapitre, le cap difficile de la théorie est passé. Asseyons-nous devant notre Amstrad et voyons comment nous allons l'obliger à comprendre puis à exécuter un programme écrit en assembleur.

4

ÉTUDE D'UN EXEMPLE

Rentrez dans votre Amstrad le programme suivant et faites-le exécuter :

```
10 MODE 2 : MEMORY 43800 : FOR I = 43801 TO 43810 :
   READ A$ : POKE I , VAL ("&H" + A$) : NEXT
20 DATA 21 , 78 , CO , 36 , FF , 23 , 23 , 36 , FF , C9
30 CALL 43801
```

Vous voyez apparaître, vers le haut de votre écran et au milieu, deux petits segments jaunes. Ces segments n'ont pu être allumés par POKE, comme dans le Chapitre 2 ; nous en retrouverions l'indication au milieu du programme BASIC. Nous les avons obtenus grâce à l'exécution de notre premier programme écrit en langage machine.

Lignes	Codes machine	Assembleur		
1	21 78 C0	LD	HL,49272	(immédiat)
2	36 FF	LD	(HL),255	(immédiat)
3	23	INC	HL	(registre)
4	23	INC	HL	(registre)
5	36 FF	LD	(HL),255	(immédiat)
6	C9	RET		(inhérent)

C'est cette présentation que nous reprendrons tout au long de ce livre. Pour l'heure, faisons l'analyse minutieuse de tout ce qui est nouveau.



PARTIE ASSEMBLEUR

C'est celle que le lecteur assimilera le plus vite.

```
LD   HL,49272
```

Nous retrouvons une instruction déjà rencontrée. Elle signifie que le registre HL va contenir la valeur décimale 49272. Décomposons ce nombre pour obtenir les poids fort et faible. Le détail de ce travail ne sera plus repris dans le reste du livre.

49272/256 = 192 (division entière)

donc le poids fort est égal à 192 (soit C0 hexadécimal).

49272 − (256 * 192) = 49272 − 49152 = 120

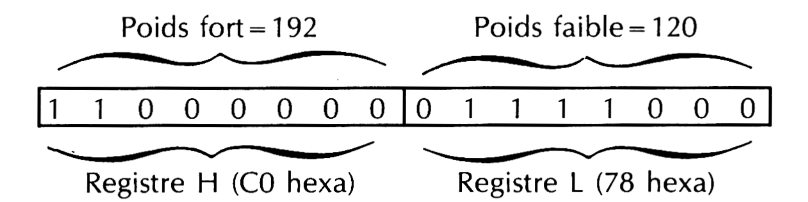
le poids faible est donc égal à 120 (soit 78 hexadécimal).
Nous obtenons l'égalité :

49272 = 192 * 256 + 120

c'est-à-dire :

Poids fort * 256 + poids faible.

Cela nous donne la configuration du registre HL :



d'où l'on déduit que les registres H et L contiennent respectivement les valeurs 192 et 120. Nous aurons besoin de ces indications un peu plus tard. En ce qui concerne notre instruction, retenons simplement que la valeur écrite dans le registre HL vaut 49272 et que le mode d'adressage utilisé est l'immédiat.

```
LD   (HL),255
```

Le nombre 255 est écrit non pas dans le registre HL mais, comme dans le cas précédent, dans l'octet pointé par HL. Puisque ce registre détient la valeur 49272, le contenu de l'octet numéro 49272 est porté à 255. Les deux premières instructions assembleur sont donc équivalentes à la ligne BASIC :

```
POKE 49272 , 255
```

Le Chapitre 2 nous a montré que l'octet 49272 se trouvait dans la zone de la mémoire écran. Il correspond au quarantième segment de la neuvième ligne de l'image. Comme le programme a été écrit sur le mode 1 et que 255 est égal à 11111111 en binaire, on com-

prend pourquoi le segment considéré voit la totalité de ses points allumés en jaune. Au sujet de la dénomination du mode d'adressage employé, certains trouveront qu'il s'agit en réalité de l'indirect. Pourquoi pas, c'est bien le contenu d'un registre qui est concerné. Mais c'est bien une valeur immédiate, 255, qui est chargée, non ? N'engageons pas le débat sur le fond de la question. L'important est de comprendre ce que réalise notre programme.

```
INC  HL
```

Le contenu du registre HL est incrémenté ; son contenu, augmenté d'une unité, passe donc à 49273. Cette instruction se contente de modifier la valeur de l'un des registres du microprocesseur. Aucun effet ne doit en être attendu à l'extérieur, sur l'écran par exemple.

```
INC  HL
```

Une nouvelle unité est ajoutée à HL. Ce registre contient maintenant la valeur 49274.

```
LD   (HL),255
```

L'effet de cette instruction est identique à celui obtenu précédemment. Le segment relié à l'octet 49274 s'allume, avec tous ses points colorés en jaune. Il est séparé du segment précédent par un espace égal à la largeur d'un caractère (en mode 2).

```
RET
```

Cette instruction, souvent écrite en dernière ligne, sert à indiquer au microprocesseur que le programme assembleur est terminé. C'est le BASIC qui reprend alors le contrôle de la situation.

CODES MACHINE

Un ordinateur ne comprend que des nombres et, pour lui, les expressions LD, INC ou RET ne veulent absolument rien dire. Il va donc falloir lui traduire le programme que nous avons écrit en assem-

bleur sous la seule forme qui lui soit compréhensible : le code machine.

Insistons bien sur la différence qu'il y a entre les langages assembleur et machine : le premier est conçu pour l'esprit humain et il est formé d'instructions qui ont un sens pour nous. Quand on écrit LD HL,49272, on sait très bien ce qui se passera dans le processeur ; on n'a pas besoin de faire un gros effort pour comprendre que le contenu du registre HL sera modifié et remplacé par une nouvelle valeur. La forme assembleur permet d'écrire des programmes qui soient lisibles, des programmes qui soient écrits avec des mots ou des abréviations dont on s'habituera vite à connaître le sens. Qui, parmi nous, serait capable d'interpréter la suite des codes machine 21, 78, C0, 36... qui, de surcroît, sont écrits en hexadécimal ?

Livrons-nous donc à la traduction en langage machine du programme. C'est un exercice plutôt simple, mais attention à la moindre erreur de codage qu'il sera ensuite très difficile de retrouver ! Servez-vous du tableau de l'Annexe D.

21 est l'équivalent pour la machine de LD. Vous constatez que LD se code de différentes façons suivant le mode d'adressage et les registres concernés. Le registre qui nous intéresse est HL et le mode d'adressage est l'immédiat. Il faudra toujours se souvenir que les codes machine sont écrits en hexadécimal ; 21, ainsi que tous les autres codes de ce tableau, respecte cette règle.

78 et **C0** sont, comme nous l'avons vu, les poids faible et fort du nombre 49272. Le microprocesseur, après avoir interprété 21, s'attendra à ce qu'on lui dise avec quel nombre il doit charger HL. Puisque 78 et C0 viennent à la suite de 21, il comprendra que la valeur C078 (ou 49272 décimal) doit être placée dans le double registre. Arrêtons-nous une minute sur une particularité du Z80. Ce processeur a l'habitude d'intervertir l'ordre logique des poids faible et fort. Pour lui, 49272 s'écrit 78 suivi de C0 et non pas le contraire. N'essayez pas de le faire changer d'avis, il aurait quand même le dernier mot. Alors autant noter ses petites manies et s'en souvenir, non ?

36 est le code machine de l'instruction LD (HL) en mode immédiat. Quand l'ordinateur lit ce code, il sait qu'il lui faut alors s'intéresser à la valeur suivante. Effectivement, si l'octet pointé par HL doit être modifié, il est nécessaire de savoir avec quel nombre.

FF est la valeur obtenue en convertissant 255 en hexadécimal. Le microprocesseur a maintenant en sa possession tous les éléments pour réaliser l'allumage du premier petit segment jaune.

23 correspond à l'instruction INC HL. La machine, rencontrant ce nombre, va exécuter l'action voulue et faire passer le contenu de HL à 49273.

23 n'a pas changé de signification pour le processeur. Il effectue une nouvelle incrémentation de son registre HL.

36 et **FF,** une fois interprétés, conduisent à l'apparition sur notre écran d'un deuxième segment. Le microprocesseur a en effet compris qu'il lui fallait écrire la valeur 255 dans l'octet 49274.

C9 est la traduction en langage machine de l'instruction RET. Ce nombre termine la série des codes machine et redonne la main au BASIC.

PROGRAMME BASIC

MEMORY est une instruction qui permet de réserver de la place en mémoire. Elle est utilisée pour indiquer à l'ordinateur qu'il ne devra pas se servir d'un certain nombre d'octets de la mémoire.

```
MEMORY 43800
```

doit être interprétée de la façon suivante : l'ordinateur pourra utiliser n'importe quel octet dont l'adresse est inférieure ou égale à 43800 mais ne devra en aucun cas modifier les contenus des octets 43801, 43802, 43803... 43903. Ce dernier nombre est l'adresse de la fin de la mémoire utilisateur de l'Amstrad.

```
FOR I = 43801 TO 43810
```

Nous avons besoin de ces dix octets et nous sommes certains, grâce à MEMORY, qu'ils seront réservés à notre usage.

```
READ A$
```

La lecture des dix valeurs écrites en DATA sera effectuée ; il est nécessaire de faire lire une chaîne de caractères car, d'une part, des

lettres apparaissent dans la ligne 20 et, d'autre part, les chiffres eux-mêmes sont écrits en hexadécimal et non en décimal.

```
POKE I , VAL ("&H" + A$)
```

Au début de la boucle, I vaut 43801 et A$ "21". L'interpréteur reconnaît donc le nombre hexadécimal &H21 et il en écrit la valeur dans l'octet 43801. Au deuxième passage, le nombre 78 sera placé dans l'octet 43802. Après le dernier passage, les dix codes machine seront inscrits dans les octets allant de 43801 à 43810.

```
NEXT
```

Voilà notre programme chargé en mémoire centrale et il ne reste plus qu'à l'exécuter.

```
CALL 43801
```

D'une manière identique à RUN qui lance un programme BASIC, CALL démarre l'exécution du programme machine. La commande des opérations passe au microprocesseur qui va réaliser les instructions correspondant aux valeurs qui se trouvent dans les octets 43801 et suivants. En rencontrant 21, il va charger le registre HL avec le nombre qui se trouve dans les deux octets d'après et continuer ainsi jusqu'à la valeur C9.

REMARQUE

On peut se demander quel est le rôle des octets dont les adresses vont de 43811 à 43903. Il est nul ; ces octets ne sont utilisés ni par l'ordinateur puisque MEMORY l'en empêche, ni par nous car, en fin de compte, notre programme n'a besoin que de dix octets. La logique aurait voulu que l'on écrive le BASIC avec les modifications suivantes :

```
10 MODE 2 : MEMORY 43893 : FOR I = 43894 TO 43903 ...
30 CALL 43894
```

La ligne 10 aurait placé dans les octets 43894 à 43903 les dix codes machine que le processeur aurait retrouvés avec CALL 43894. Nous avons choisi de bloquer tous les octets à partir de 43801 pour une raison bien simple : la presque totalité des programmes de ce livre sera logée en mémoire à partir de cette adresse afin d'avoir une présentation uniforme. Et ce premier exemple respecte cette règle. Ajoutons que la perte d'une centaine d'octets ne doit pas être considérée comme excessive et qu'il est tout de même plus facile, si l'on a un programme de 15 octets par exemple, d'écrire

```
FOR I = 43801 TO 43815
```

plutôt que

```
FOR I = 43889 TO 43903
```

UTILISATION DE L'ÉDITEUR/ASSEMBLEUR ZEN

Si vous avez fait l'acquisition de la cassette qui permet de programmer directement l'Amstrad en assembleur, ce paragraphe doit pouvoir guider vos premiers pas.

Placez la cassette dans le lecteur associé à votre ordinateur et tapez la ligne BASIC suivante :

```
MEMORY 16383 : LOAD "ZEN"          (Enter)
```

Le magnétophone se met alors en marche et le programme écrit sur la cassette est transféré en mémoire centrale. C'est ce programme qui va avoir pour tâche de traduire (à notre place) les commandes assembleur en codes machine. Nous allons pouvoir, sous le contrôle de ce programme, taper directement sur le clavier une instruction comme par exemple LD HL,49272. L'ordinateur nous fournira automatiquement les trois codes machine correspondants : 21, 78 et C0 et, de plus, il les écrira dans les octets de la mémoire que nous aurons choisis.

Commençons par le commencement.

L'instruction MEMORY 16383 nous est maintenant familière. Elle va empêcher le BASIC d'utiliser les octets dont l'adresse est supérieure à 16383. Cela est nécessaire car le programme provenant de la cassette est justement chargé en mémoire à partir de l'octet numéro 16384. Dès que ce chargement est effectué, on abandonne le BASIC :

```
CALL 16384        (Enter)
```

Le mot ZEN suivi du signe > est alors visible sur le téléviseur et l'ordinateur attend nos ordres.

Appuyons sur Enter pour que l'écran s'efface et enfonçons successivement les touches E et Enter. Immédiatement un numéro de ligne apparaît. Il n'y a plus qu'à entrer un programme en appuyant sur Enter à la fin de chaque ligne.

```
1   ORG    43801
2   LOAD   43801
3   LD     HL,49272
4   LD     (HL),255
5   INC    HL
6   INC    HL
7   LD     (HL),255
8   RET
9   END
10  .
```

Vous aurez reconnu le programme étudié depuis le début de ce chapitre. Quatre lignes ont toutefois été ajoutées ; les voici :

```
1   ORG    43801
2   LOAD   43801
9   END
10  .
```

Les deux premières indiquent à l'ordinateur qu'il doit générer les codes machine et les écrire dans des octets successifs de la mémoire. Le premier de ceux-ci aura pour adresse 43801.

Les deux dernières servent à délimiter la fin du programme. Quand le point (suivi de Enter) de la ligne 10 aura été tapé, vous retrouverez le message ZEN > .

Appuyez alors sur les touches A (puis Enter) et V (puis Enter). Pendant un court instant, l'ordinateur procède à l'assemblage de notre programme : nos instructions assembleur sont alors traduites en langage machine. Dès que ce travail est terminé, on voit apparaître sur l'écran le listing d'assemblage suivant :

```
                    ORG   43801
                    LOAD  43801
AB19    2178C0      LD    HL,49272
AB1C    36FF        LD    (HL),255
AB1E    23          INC   HL
AB1F    23          INC   HL
AB20    36FF        LD    (HL),255
AB22    C9          RET
                    END
```

La partie droite est celle que nous avons entrée. La partie centrale correspond aux codes machines 21, 78, C0, 36, etc. Et la colonne de gauche n'est pas autre chose que la traduction hexadécimale des nombres compris entre 43801 et 43810.

Résumons : le programme ZEN a traduit les instructions assembleur en une série de codes machine et a inscrit ces codes dans les 10 octets 43801, 43802... 43810. OK ? Il nous reste à comprendre comment utiliser ces données. Tapez B (pour Bye) puis Enter. Vous repassez à cet instant sous le contrôle du BASIC.

```
10 MODE 2 : CALL 43801
```

Faites exécuter ce court programme. Vous verrez apparaître deux segments jaunes sur votre écran. L'instruction CALL a envoyé le microprocesseur exécuter les codes machine écrits à partir de l'adresse 43801. Ces codes ont, comme on le sait, été placés là par le programme ZEN et n'ont pas subi de modification quand le BASIC a été rappelé.

Une dernière chose pour finir : il est tout à fait possible de retrouver notre programme assembleur, le retour au BASIC ne l'ayant pas effacé de la mémoire de l'Amstrad. La marche à suivre est donnée ici :

```
CALL 16384          (Enter)
Le message ZEN >  apparaît
Tapez Pn            (Enter)
```

P est l'abréviation de PRINT et n est le nombre de lignes que l'on souhaite faire écrire. Dans notre exemple, il faudra taper P9 puisque notre programme est long de 9 lignes. La totalité du listing sera alors affichée sur l'écran.

5

ÉLÉMENTS DE PROGRAMMATION DU Z80

Note importante

Le lecteur ne devra pas oublier, dans chacun des programmes qui sont donnés à titre d'exemples, d'inclure les lignes 10 et 20 suivantes :

```
10 MEMORY 43800 : FOR I = 43801 TO 438?? : READ A$ :
   POKE I , VAL ( "&H" + A$ ) : NEXT
20 DATA ...
```

Les deux points d'interrogation de la ligne 10 devront être remplacés par le nombre d'octets du programme en langage machine et la ligne 20 devra contenir en DATA la liste des codes machine écrits sous forme hexadécimale.

LD A

Abréviation de LoaD A *(charger A), cette instruction permet de placer une valeur 8 bits dans le registre A.*

Exemple : MODE D'ADRESSAGE IMMÉDIAT (11 octets)

Programme BASIC

```
 5 MODE 1 : INK 2,6 : INK 3,11,16
10 MEMORY 43800 : FOR I = 43801 TO 43811 : READ A$ :
   POKE I , VAL ( "&H" + A$ ) : NEXT
20 DATA 3E , 0F , 32, 28 , C0 , 3E , FF , 32 , 78 , C0 , C9
30 CALL 43801
```

Les lignes 10 et 20 sont données dans cet exemple mais seuls leurs numéros apparaîtront dans le reste de ce livre. Pensez à chaque fois à les compléter.

Programme assembleur

Lignes	Codes machine	Assembleur	
1	3E 0F	LD	A,15
2	32 28 C0	LD	(49192),A
3	3E FF	LD	A,255
4	32 78 C0	LD	(49272),A
5	C9	RET	

La présentation du programme sous forme de tableau n'est faite que dans un souci de clarté. Contrairement au BASIC, le numéros

de ligne n'ont aucune importance pour le microprocesseur qui ne s'intéresse, dans cet exemple, qu'à la suite des octets 43801 à 43811.

Ligne 1 : la valeur 15 (0F hexadécimal ou 00001111 binaire) est mise dans l'accumulateur A. 3E est le code hexadécimal de l'instruction LD A en mode immédiat.

Ligne 2 : le contenu du registre A est rangé dans l'octet 49192. 32 est le code machine de l'instruction, 28 (40 décimal) et C0 (192 décimal) sont les poids faible et fort du nombre 49192.

49192 = 192 * 256 + 40

Rappelez-vous que le microprocesseur s'occupe toujours du poids faible en premier.

Dès que les lignes 1 et 2 sont exécutées, un premier segment apparaît ; il est situé au milieu de la première ligne de l'écran et chacun de ses huit points est coloré en rouge. Voici ce qui avait été dit au sujet des octets de la mémoire vidéo en mode 1 :

	7	6	5	4	3	2	1	0	← numéros des bits
Octet 49192 →	0	0	0	0	1	1	1	1	

Les bits 3 et 7 (dans cet ordre) donnent la couleur du premier quart gauche du segment. L'analyse de leur contenu fournit la valeur binaire 10 (2 décimal). En mettant cela en parallèle avec le fait que la ligne BASIC 5 contient l'instruction INK 2,6 on en déduit la teinte de ce premier quart : rouge.

La même étude peut se réaliser sur les trois autres quarts du segment et l'on comprend alors pourquoi la totalité de celui-ci apparaît en rouge.

Ligne 3 : une nouvelle valeur est inscrite dans l'accumulateur. Les traductions hexadécimale et binaire de 255 sont respectivement FF et 11111111.

Ligne 4 : la valeur 255 se retrouve dans l'octet numéro 49272. Comme l'écart entre 49272 et 49192 est égal à 80, on va voir se dessiner sur l'image un deuxième segment placé exactement au-dessous du précédent et séparé de lui de la hauteur d'une case caractère. Regardons la configuration de l'octet 49272 :

Octet 49272 → | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Les couples de bits (3,7), (2,6), (1,5) et (0,4) fournissent la même valeur binaire 11 (3 décimal). Comme la troisième couleur d'encre est donnée par la commande INK 3,11,16 on peut en tirer la conclusion suivante : le segment doit nous apparaître en clignotant, prenant alternativement les couleurs bleu ciel (teinte numérotée 11) et rose (teinte numérotée 16).

Ligne 5 : C9 est le code de l'instruction RET qui indique que le programme machine est terminé. On retourne alors au BASIC.

LD (IX+n)

Cette instruction permet d'écrire une valeur 8 bits dans l'octet dont l'adresse est calculée en ajoutant au contenu du registre IX la valeur n.

Exemple : MODE D'ADRESSAGE INDIRECT (13 octets)

Programme BASIC

```
 5 MODE 2
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur	
1	DD 21 3B C0	LD	IX,49211
2	DD 36 14 AA	LD	(IX+20),170
3	DD 36 EC AA	LD	(IX−20),170
4	C9	RET	

Ligne 1 : le registre IX est chargé, en mode immédiat, avec la valeur 49211. Il conservera ce contenu durant tout le programme. Au sujet du codage machine, notez que l'instruction LD correspond à elle seule aux valeurs hexadécimales DD et 21. Nous aurons l'occasion de retrouver d'autres instructions assembleur devant se coder sur deux octets. Quant à la transcription du nombre 49211, faut-il encore rappeler qu'elle se fait en deux parties, poids faible (59 décimal ou 3B hexa) puis poids fort (192 décimal ou C0 hexa) ?

Ligne 2 : puisque le registre IX contient le nombre 49211, il est facile d'en déduire que IX + 20 a pour valeur 49231. Ce résultat est l'adresse du dernier segment de la première ligne de l'écran, plus exactement de l'octet qui lui est rattaché. Comme on y écrit 170, on va donc voir apparaître notre segment à l'extrême droite du téléviseur. En mode 2, l'utilisateur ne dispose que de deux couleurs ; en supposant que ce sont les couleurs standard (jaune et bleu) et en supposant que la qualité de notre vue est irréprochable, nous devons être capables de distinguer que les huit points du segment prennent alternativement les couleurs jaune et bleu. Cela est explicable par le fait que 170 s'écrit 10101010 en binaire.

Un mot du codage de la commande LD (IX+20),170 :

DD et 36 sont les codes machine de l'instruction elle-même ;
14 est la notation hexadécimale de 20 ;
AA est celle de 170. OK ?

Ligne 3 : nous reprenons le registre IX et nous lui soustrayons 20. Le résultat est égal à 49191. Le contenu de l'octet dont on vient d'obtenir l'adresse est porté à 170 et un deuxième segment, situé à peu près au milieu de l'écran et sur la première ligne, est rendu visible. Il a la même structure que dans le cas précédent : un point sur deux est allumé en jaune.

Remarquons, pour finir, que le nombre négatif −20 a été codé sous la forme "complément à deux sur 8 bits".

20 décimal :	00010100
inversion des chiffres :	11101011
ajout de 1 :	11101100 (soit 236 décimal ou EC hexadécimal).

LD A,B

Cette instruction programme la recopie dans l'accumulateur du contenu du registre B. Ce dernier registre n'est pas modifié.

Exemple : MODE D'ADRESSAGE REGISTRE (12 octets)

Programme BASIC

```
 5 MODE 0
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur	
1	01 35 EE	LD	BC,60981
2	78	LD	A,B
3	32 E6 C3	LD	(50150),A
4	79	LD	A,C
5	32 E8 C3	LD	(50152),A
6	C9	RET	

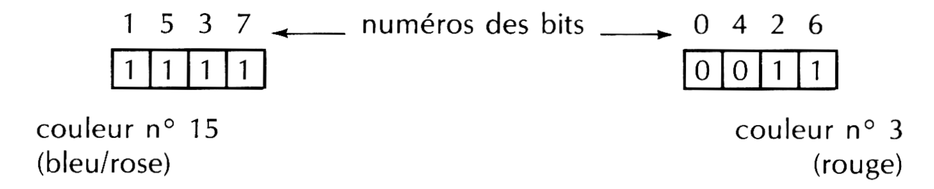
Ligne 1 : on écrit dans le registre 16 bits BC, en mode immédiat, la valeur 60981. La décomposition de ce nombre en poids fort et faible donne :

Registre B : 238 (ou EE hexadécimal)
Registre C : 53 (ou 35 hexadécimal)

Ligne 2 : l'accumulateur est chargé avec la valeur que contient le registre B. On ignore quel était le contenu de A avant cette instruction, mais maintenant on est sûr d'une chose : ce contenu est égal à 238.

Ligne 3 : 50150 est l'adresse d'un octet de la mémoire écran ; il correspond à un segment disposé à peu près au milieu du téléviseur. Pour savoir avec quelles teintes il va être rendu visible, il est nécessaire de rassembler nos souvenirs :

En mode 2, 16 couleurs sont disponibles. Mais chaque segment ne peut prendre que deux couleurs. Examinons la structure binaire de 238 — soit 11101110 — et mélangeons tous ces chiffres de la même façon que le fait notre ordinateur.



Puisque, dans le programme BASIC, nous n'avons pas modifié les couleurs des 16 encres dont dispose l'Amstrad à l'initialisation, nous voyons apparaître le segment 50150 avec les teintes bleu ciel sur rose (clignotant) d'une part, et rouge (fixe) d'autre part.

Vous aurez certainement remarqué que les lignes 2 et 3 auraient pu être avantageusement remplacées par la seule commande :

```
LD (50150),B
```

Hélas, l'écriture de la valeur du registre B directement dans un octet de la mémoire n'est pas chose faisable par le Z80. Ce ne sera pas la seule fois que nous aurons la possibilité de constater que l'accumulateur A joue un rôle privilégié au sein du microprocesseur.

Lignes 4 et 5 : on transfère le contenu de C dans A. Les registres A et C ont maintenant la même valeur : 53 décimal ou 00110101 binaire. Cette valeur est transmise à l'octet 50152 et un segment, placé légèrement à droite de celui qui est déjà dessiné, va s'éclairer. Il nous apparaîtra en blanc (partie gauche) et en clignotant bleu/jaune (partie droite).

1	5	3	7
0	1	0	0

← numéros des bits →

0	4	2	6
1	1	1	0

couleur n° 4
(blanc)

couleur n° 14
(bleu/jaune)

LD — Résumé

Voici les différentes possibilités offertes au programmeur dans l'utilisation de l'instruction LD (LOAD).

LD A,n8 LD B,n8 LD C,n8 LD D,n8 LD E,n8 LD H,n8 LD L,n8	LD BC,n16 LD DE,n16 LD HL,n16 LD SP,n16 LD IX,n16 LD IY,n16
n8 est un nombre de 8 bits (registre simple) valant 255 au maximum. n16 est un nombre de 16 bits (registre double) valant 65535 au maximum.	

Exemple :

```
LD  A,25        A contient la valeur 25
LD  H,50        H contient la valeur 50
LD  DE,40000
```

DE contient la valeur 40000. Cette instruction est tout à fait équivalente des deux lignes suivantes :

```
LD  D,156       (poids fort de 40000)
LD  E,64        (poids faible de 40000)
```

LD R,R'	LD SP,HL
R et R' sont n'importe lesquels des registres A, B, C, D, E, H ou L.	LD SP,IX LD SP,IY

Exemple :

```
LD   B,100
LD   H,B
LD   L,B
LD   SP,HL
```

L'exécution de ce petit programme place dans B, H et L la même valeur décimale 100. On en déduit alors la valeur commune des registres HL et SP : 100 * 256 + 100 = 25700.

LD R,(HL)	LD (HL),R
R est n'importe lequel des registres 8 bits A, B, C, D, E, H ou L.	

Exemple :

```
LD   B,50
LD   HL,20000
LD   (HL),B
LD   C,(HL)
```

Ce programme écrit dans B le nombre 50, dans HL le nombre 20000, dans l'octet numéro 20000 le nombre 50 et dans le registre C le nombre 50. D'accord ?

LD A (adresse) LD A,(BC) LD A,(DE)	LD (adresse),A LD (BC),A LD (DE),A
L'accumulateur est le seul registre 8 bits à pouvoir correspondre avec un octet de la mémoire par l'intermédiaire d'une adresse (adressage étendu) ou des pointeurs registre BC et DE (adressage indirect)	

Exemple :

```
LD  BC,60000
LD  A,(BC)
INC A
LD  (BC),A
```

Première et deuxième lignes : BC est chargé avec le nombre 60000 et A avec le contenu de l'octet 60000.

Troisième et quatrième lignes : ce contenu est incrémenté et réécrit dans l'octet 60000. Ce dernier aura donc vu sa valeur augmentée d'une unité.

LD (n16),Rd	LD Rd,(n16)
n16 est un nombre 16 bits et Rd est un registre double quelconque parmi les suivants : BC, DE, HL, SP, IX ou IY.	

Exemple :

```
LD  DE,30000
LD  (10000),DE
```

En inscrivant 30000 dans DE, on attribue au registre D la valeur 117 (poids fort) et au registre E la valeur 48 (poids faible). Naturellement, le contenu du registre DE ne risque pas d'entrer dans le seul octet numéro 10000, le suivant devra donc être utilisé. N'oubliez pas que le microprocesseur écrit le poids faible d'abord — 48 dans l'octet 10000 — et le poids fort ensuite — 117 dans l'octet 10001.

LD (HL),n8 LD (IX+n),n8 LD (IY+n),n8	
Le nombre 8 bits n8 est écrit dans l'octet dont l'adresse est contenue par HL ou déduite de IX ou IY.	

Exemple :

```
LD  IX,20000
LD  (IX+5),100
```

La valeur 100 se retrouve dans l'octet d'adresse 20005. Notons que si n est négatif, on doit le noter sous la forme "complément à 2".

LD (IX+n),R LD (IY+n),R	LD R,(IX+n) LD R,(IY+n)
R est l'un quelconque des registres 8 bits A, B, C, D, E, H ou L.	

Exemple :

```
LD  C,150
LD  IY,5000
LD  (IY+10),C
LD  E,(IY+20)
```

C et IY étant respectivement chargés avec les nombres 150 et 5000, on en déduit que l'octet 5010 voit son contenu porté à 150. Quant au registre E, il se retrouve avec la valeur de l'octet 5020 (ce programme ne dit d'ailleurs pas quelle est cette valeur).

Prenez le temps, avant de passer à l'instruction suivante, de traduire en codes machine les petits exemples que l'on vient de donner ; et exécutez les programmes correspondants. Ne perdez pas de vue que la fonction BASIC PEEK donne le contenu de n'importe quel octet de la mémoire. Servez-vous-en pour vérifier que le microprocesseur a bien écrit le bon nombre dans le bon octet.

CALL

Cette instruction appelle un sous-programme et indique au microprocesseur à quelle adresse le sous-programme doit démarrer. Le retour s'effectue à l'instruction qui suit CALL.

Exemple : MODE D'ADRESSAGE ABSOLU (11 octets)

Programme BASIC

```
 5 MODE 0 : LOCATE 10 , 10
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur
1	3E 41	LD A,65
2	CD 5D BB	CALL 47965
3	3E 42	LD A,66
4	CD 5D BB	CALL 47965
5	C9	RET

Voici un programme clé pour notre sujet. D'une part, il va nous permettre d'approfondir nos connaissances sur la façon dont fonctionne notre ordinateur ; d'autre part, il va nous faire comprendre comment nous pouvons, en assembleur, imprimer des lettres sur l'écran.

Ligne 1 : l'accumulateur est chargé avec un nombre qui est le code ASCII de la lettre majuscule A.

Ligne 2 : c'est le cœur du programme. On peut comparer CALL à l'instruction BASIC GOSUB : le microprocesseur va partir exécuter les codes machine qui se trouvent à partir de l'adresse 47965 (BB5D hexa). Vous avez du mal à comprendre, vous ne voyez aucune signification à ces nombres, vous trouvez que ce n'est pas clair ? Disons-le tout net : dans l'état d'avancement de notre étude, cela ne peut pas être clair. C'est même le trou noir. On ignore quel genre d'instructions l'ordinateur va aller exécuter ! Alors, que doit-on retenir de tout cela ?

- Premièrement, que le nombre 47965 n'a pas été choisi au hasard ; il fait partie d'une zone mémoire de la machine que nous avons dite réservée au système. Vous vous y réintéresserez plus tard, la documentation fournie avec l'Amstrad vous sera utile à ce moment-là.
- Deuxièmement, qu'à partir du moment où un programme se branche à cette adresse une lettre apparaît à l'image. Son emplacement est donné par l'instruction LOCATE de la ligne BASIC 5 : c'est approximativement le milieu de l'écran.
- Troisièmement, que le caractère affiché est en relation directe avec le contenu du registre A. Le sous-programme d'affichage fait toujours apparaître le caractère dont le code ASCII se trouve dans l'accumulateur. C'est pour cela que la lettre A est présente sur l'écran.
- Quatrièmement, qu'après avoir réalisé le travail, le processeur retrouve l'instruction qui suit immédiatement CALL. Dans notre cas, c'est de la ligne 3 qu'il s'agit.

Lignes 3 et 4 : on renouvelle l'opération et l'on voit apparaître cette fois la lettre B à côté de la précédente. Il est facile de voir que, là, le registre A a été chargé avec le code ASCII de la deuxième lettre de l'alphabet préalablement à l'appel de la routine.

Finissons notre étude par deux remarques. D'une part, l'instruction CALL s'adresse directement à un octet de la mémoire et n'utilise aucun registre. C'est pour cette raison que nous ne retrouvons pas un mode connu et que nous parlons d'adressage absolu. D'autre part, le sous-programme d'affichage déplace toujours automatiquement le curseur d'une case sur la droite. Cela explique que la lettre B soit écrite à côté de la lettre A.

ADD A

Cette instruction va ajouter les contenus du registre A et d'un octet mémoire. Le résultat de l'addition sera alors placé dans A.

Exemple : MODE D'ADRESSAGE INDIRECT (14 octets)

Programme BASIC

```
10
20
30 INPUT " PREMIER NOMBRE " ; N : POKE 43850 , N
40 INPUT " DEUXIEME NOMBRE " ; M : POKE 43851 , M
50 CALL 43801
60 PRINT " LA SOMME VAUT " ; PEEK (43852) : GOTO 30
```

Programme assembleur

Lignes	Codes machine	Assembleur	
1	DD 21 4A AB	LD	IX,43850
2	DD 7E 00	LD	A,(IX+0)
3	DD 86 01	ADD	A,(IX+1)
4	DD 77 02	LD	(IX+2),A
5	C9	RET	

Ligne 2 : le registre A est chargé avec la valeur contenue dans l'octet pointé par IX, c'est-à-dire l'octet 43850. La ligne BASIC 30 a écrit dans cet octet le premier terme de la somme. N ne doit pas, bien sûr, dépasser 255, sinon le message *improper argument* apparaîtrait sur votre écran.

Ligne 3 : ADD A,(IX+1) signifie que l'on ajoute la valeur de A et la valeur inscrite dans l'octet ayant pour adresse IX+1. De ce fait cette ligne additionne le premier et le second nombre de la somme, second nombre qui a été placé par POKE dans l'octet 43851 (ligne BASIC 40).

Ligne 4 : le résultat de l'opération étant dans l'accumulateur, il ne reste plus qu'à ranger la somme obtenue dans l'octet 43852 puisque c'est là que le BASIC ira chercher la réponse.

Faisons tourner le programme.

Premier nombre	Deuxième nombre	Somme
10	20	30
100	0	100
200	60	4
250	250	244

Il n'y a rien à redire pour les deux premiers cas : les résultats sont conformes à nos prévisions. Quant aux calculs suivants, il ne sera pas bien compliqué d'établir leur cohérence : puisque A est un registre 8 bits, le nombre le plus grand que l'on puisse y écrire est 11111111 (255 décimal). Si, à ce moment-là, on essaie d'ajouter 1, le registre repassera à 00000000 (0 décimal) et c'est ce qui explique que 260 soit devenu 4 dans notre troisième somme. D'une manière analogue, en ajoutant 250 et 250, on ne trouve pas 500 mais 500 − 256, soit 244.

DIFFÉRENTES FORMES DE L'ADDITION 8 BITS

ADD	A,n8	n8 est un nombre 8 bits.
ADD	A,R	R est l'un des registres A, B, C, D, E, H ou L.
ADD ADD ADD	A,(HL) A,(IX+n) A,(IY+n)	

ADD 16 bits

ADD permet aussi d'ajouter les valeurs inscrites dans deux registres doubles. On pourra donc additionner deux valeurs 16 bits. Le seul mode d'adressage possible est l'adressage registre.

Exemple : MODE D'ADRESSAGE REGISTRE (12 octets)

Programme BASIC

```
10
20
30 INPUT " PREMIER NOMBRE " ; N
40 N1 = INT ( N / 256 ) : POKE 43851 , N1
50 N2 = N - N1 * 256 : POKE 43850 , N2
60 INPUT " DEUXIEME NOMBRE " ; M
70 M1 = INT ( M / 256 ) : POKE 43853 , M1
80 M2 = M - M1 * 256 : POKE 43852 , M2
90 CALL 43801
100 PRINT " LA SOMME VAUT " ; 256 * PEEK (43855) + PEEK (43854)
```

Programme assembleur

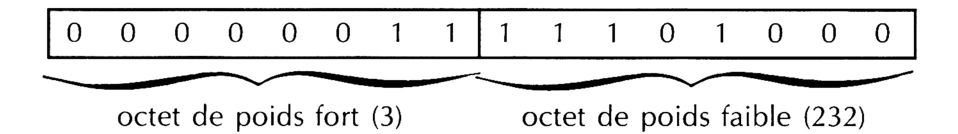
Lignes	Codes machine	Assembleur	
1	ED 4B 4A AB	LD	BC,(43850)
2	2A 4C AB	LD	HL,(43852)
3	09	ADD	HL,BC
4	22 4E AB	LD	(43854),HL
5	C9	RET	

Ligne 1 : le registre BC est chargé avec le premier nombre N. Tenant compte du fait que BC est un registre 16 bits, l'instruction LD va chercher en mémoire la valeur des 2 octets 43851 (poids fort) et 43850 (poids faible). Regardons, en supposant N égal à 1000, comment cela se passe.

N1 = INT (1000/256) soit N1 = 3. L'octet 43851 contient le nombre 3.

N2 = 1000 − 3 * 256 soit N2 = 232. L'octet 43850 contient le nombre 232.

Après exécution de la ligne, BC se présente sous la forme suivante :



En appliquant, à titre de vérification, la formule

256 * poids fort + poids faible

on obtient bien

256 * 3 + 232, c'est-à-dire le nombre 1000.

Lignes 2 et 3 : l'opération est effectuée. Le premier nombre N se trouve déjà dans BC et le second, M, doit être recherché en mémoire. Choisissons, par exemple, M égal à 2000. Les lignes BASIC 70 et 80 ont placé la valeur de M dans les octets 43852 et 43853, de la façon suivante : la partie de poids fort de 2000, c'est-à-dire 7 (M1), est inscrite à l'adresse 43853 et sa partie faible, 208 (M2), à l'adresse 43852.

On peut résumer les lignes 1, 2 et 3 en disant que tout s'est passé comme si l'on avait ajouté directement les nombres N et M écrits aux adresses 43850 et 43851 d'une part, 43852 et 43853 d'autre part. Il n'existe malheureusement pas d'instruction qui le fasse de manière directe et le passage par les registres BC et HL a été obligatoire.

Ligne 4 : le résultat de l'addition ayant été mis dans HL, il ne reste plus qu'à ranger le contenu de ce registre dans un emplacement mémoire où la ligne BASIC 100 pourra le chercher. Puisque ce résultat

est un nombre de 16 bits, l'instruction LD va le placer sur deux octets, à savoir partie faible à l'adresse 43854 et partie forte à l'adresse 43855.

En faisant exécuter le programme sur quelques exemples, vous remarquerez qu'à chaque fois que la somme dépasse 65535, le registre HL la fait repartir à zéro. 65535 est en effet la valeur décimale la plus grande que l'on puisse écrire sur 16 bits.

PREMIER NOMBRE ? 50000
DEUXIEME NOMBRE ? 20000
LA SOMME VAUT 4464, soit 70000 – 65536.

DIFFÉRENTES FORMES DE L'ADDITION 16 BITS

ADD	HL,Rd	Rd est l'un des registres doubles BC, DE, HL ou SP.
ADD	IX,Rd	Rd est l'un des registres doubles BC, DE, IX ou SP.
ADD	IY,Rd	Rd est l'un des registres doubles BC, DE, IY ou SP.

SUB 8 bits

SUB est une instruction qui permet de retrancher au contenu de l'accumulateur A une valeur 8 bits. Le résultat de l'opération est écrit dans A. Les modes d'adressage immédiat, registre et indirect sont utilisables.

Exemple : MODE D'ADRESSAGE INDIRECT (14 octets)

Programme BASIC

```
10
20
30 INPUT " PREMIER NOMBRE " ; N : POKE 43850 , N
40 INPUT " DEUXIEME NOMBRE " ; M : POKE 43851 , M
50 CALL 43801
60 PRINT " LA DIFFERENCE VAUT " ; PEEK (43852)
```

Programme assembleur

Lignes	Codes machine	Assembleur	
1	DD 21 4A AB	LD	IX,43850
2	DD 7E 00	LD	A,(IX+0)
3	DD 96 01	SUB	(IX+1)
4	DD 77 02	LD	(IX+2),A
5	C9	RET	

Ligne 1 : le registre IX est chargé en mode immédiat avec le nombre 43850. Il faut mettre en avant le fait que le premier terme de la différence est justement rangé à cette adresse-là, et que le second a été rangé dans l'octet suivant.

Ligne 2 : LD A,(IX + 0) signifie que l'on place dans l'accumulateur le nombre qui se trouve dans l'octet 43850 (IX + 0) : c'est la ligne BASIC 30 qui a auparavant inscrit dans cet octet le nombre N.

Ligne 3 : on retranche à N le nombre M contenu dans l'octet 43851 (IX + 1). Le nombre M est connu du programme dès que la ligne BASIC 40 est exécutée.

Notons que la soustraction se fait toujours dans le même sens : c'est l'octet mémoire qui est retranché à l'accumulateur et non le contraire. De plus on remarquera que l'écriture de l'instruction ne fait pas apparaître le nom du registre A. Cela serait inutile car le seul registre par lequel transitent toutes les soustractions est précisément A.

Ligne 4 : puisque le résultat de l'opération est maintenant dans A, il ne reste plus qu'à stocker ce résultat dans un emplacement mémoire déterminé que le programme BASIC pourra retrouver. La ligne 60 commande d'aller chercher la réponse dans l'octet 43852 (IX + 2).

DIFFÉRENTES FORMES DE LA SOUSTRACTION 8 BITS

SUB	R	Différence effectuée entre l'accumulateur et l'un quelconque des registres A, B, C, D, E, H ou L.
SUB SUB SUB	(HL) (IX+n) (IY+n)	Différence effectuée entre l'accumulateur et l'octet mémoire pointé.
SUB	n8	Différence effectuée entre l'accumulateur et le nombre n8.

Notes :

Le résultat d'une soustraction se retrouve toujours dans l'accumulateur.

La soustraction entre deux valeurs 16 bits n'est pas possible avec l'instruction SUB.

JR Z JR NZ JR CALL

Tous les programmes que nous avons étudiés jusqu'à maintenant étaient conçus sur le type séquentiel, ce qui veut dire que les instructions étaient exécutées les unes après les autres, dans l'ordre où elles avaient été écrites. Nous savons tous qu'en BASIC il est possible, avec des instructions comme GOTO par exemple, d'empêcher le programme de se dérouler normalement en l'obligeant à se brancher à un numéro de ligne choisi par nous. Voyons comment nous devrons nous y prendre pour obtenir le même effet. Nous retrouverons, après quelques explications théoriques, l'étude d'exemples bien concrets.

JR Z

Branchement si zéro

Le branchement à l'une des parties du programme machine ne se fera que si l'une des deux conditions suivantes vient d'être réalisée :

1. Résultat d'une opération égal à zéro.
2. Comparaison entre deux nombres égaux.

C'est l'octet qui suit immédiatement l'instruction JR Z qui, en mode complément à deux, indiquera alors au microprocesseur quelle autre partie du programme devra être exécutée.

Dans le cas d'une comparaison entre deux nombres différents ou d'une opération ne donnant pas zéro, JR Z n'aura aucun effet et c'est l'instruction écrite à la ligne d'après qui sera exécutée.

Pour résumer, disons que l'instruction JR Z s'utilise de la même façon que la phrase BASIC bien connue :

```
IF A ± B = 0 THEN ...
```

JR NZ

Branchement si non zéro

Voici l'instruction contraire de la précédente. Cette fois, le branchement ne sera effectué que dans le cas où l'une des deux situations suivantes se sera présentée :

1. Résultat d'une opération non égal à zéro.
2. Comparaison effectuée sur deux nombres différents.

Ici aussi, l'endroit du programme où le branchement devra se faire sera indiqué par l'octet placé après l'instruction JR NZ. Le nombre écrit dans cet octet devra l'être sous la forme complément à deux.

On peut trouver l'équivalent BASIC de JR NZ en écrivant :

```
IF A ± B < > 0 THEN ...
```

JR

Branchement dans tous les cas

Le branchement à la partie du programme indiquée par l'octet qui suit l'instruction JR est un branchement inconditionnel. Ce type de branchement ne se préoccupe pas de savoir si telle ou telle condition a été réalisée : il s'effectue de toute manière.

Vous aurez reconnu en JR l'équivalent assembleur de la commande BASIC GOTO.

CALL

Branchement vers un sous-programme

Après GOTO, voici GOSUB : CALL est en effet l'instruction de branchement qui permet de sauter jusqu'à un sous-programme. Il s'agit,

comme pour JR, d'un branchement inconditionnel qui s'effectuera dans tous les cas.

Il est inconcevable, en BASIC, d'écrire un GOSUB sans prévoir le RETURN qui nous ramènera au programme principal. Il en est de même en assembleur et il nous faudra toujours penser à terminer nos sous-programmes par une instruction que nous avons rencontrée dès notre premier exemple : RET.

INC A

Cette instruction incrémente le registre A, c'est-à-dire qu'elle lui ajoute une unité. On l'emploie sur le mode d'adressage registre.

Exemple : MODE D'ADRESSAGE REGISTRE (15 octets)

Programme BASIC

```
 5 MODE 2 : LOCATE 1 , 10
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur
1	21 00 C0	LD HL,49152
2	11 02 00	LD DE,2
3	3E 00	LD A,0
4	36 FF	SUITE: LD (HL),255
5	19	ADD HL,DE
6	3C	INC A
7	20 FA	JR NZ,SUITE (−6)
8	C9	RET

Lignes 1, 2 et 3 : l'initialisation du programme se réalise en écrivant dans HL la valeur 49152, dans DE la valeur 2 et dans A la valeur 0. Le registre HL contient l'adresse du premier octet de la mémoire vidéo — celui qui correspond au segment situé en haut et tout à gauche du téléviseur.

Ligne 4 : on attribue la valeur 255 à l'octet pointé par HL. Le segment qui lui est relatif s'allume avec ses huit points colorés en jaune (255 = 11111111 binaire).

Ligne 5 : on ajoute les contenus des doubles registres HL et DE. Ce qui revient à ajouter 2 unités à HL. Il passe alors à 49154.

Ligne 6 : l'accumulateur est incrémenté ; sa nouvelle valeur est donc 1.

Ligne 7 : nous voici devant notre première instruction de branchement ; aussi prenons tout notre temps.

JR NZ,SUITE doit s'interpréter de la façon suivante : le programme retournera exécuter la ligne 4 si le résultat de l'opération précédente est différent de zéro (s'il est Non Zéro donc). Or, quelle est l'opération de la ligne 6 ? Une addition. Et quel est le résultat de cette addition ? 1, n'est-ce pas ? Alors, que fera le microprocesseur ? Il repartira exécuter la ligne 4.

Là, il mettra à 1 les huit bits de l'octet pointé par HL — octet 49154. Un deuxième segment s'allumera alors sur l'écran ; il sera séparé du précédent par la largeur d'une case caractère en mode 2. Puis le microprocesseur ajoutera 2 unités au registre HL (ligne 5) et incrémentera l'accumulateur. Le résultat de cette opération, ici encore différent de zéro, rebranchera l'ordinateur à la ligne 4. Et un troisième segment sera dessiné.

Le principe de la boucle doit maintenant être compris ; il ne reste qu'à voir les détails :

- Le programme arrête de boucler quand l'instruction INC A de la ligne 6 donne un résultat nul ; cela se produit quand l'accumulateur, une fois arrivé à 255, doit repasser à 0. 256 segments sont alors visibles sur l'écran, séparés les uns des autres de la grandeur d'une case caractère (aussi bien en largeur qu'en hauteur).
- Il faut s'habituer à la manière dont est présentée la partie assembleur. La ligne 4 a été appelée SUITE et du coup la ligne 7 s'écrit JR NZ,SUITE au lieu de s'écrire JR NZ, ligne 4. JR NZ,SUITE a été codée 20 FA. 20 est le code machine de l'instruction JR NZ et FA est en complément à 2 le nombre −6. On indique ainsi au microprocesseur qu'il doit repartir en arrière de 6 octets, le décompte s'effectuant toujours à partir du premier octet de la ligne qui suit l'instruction de branchement.

0 ⟶ C9 ; −1 ⟶ FA ; −2 ⟶ 20 ...... −6 ⟶ 36

(36 étant le premier octet de la ligne à laquelle le branchement doit se faire).

DIFFÉRENTES FORMES DE L'INCRÉMENTATION D'UN REGISTRE

INC	R	R est l'un des registres A, B, C, D, E, H ou L.
INC	Rd	Rd est l'un des registres BC, DE, HL, SP, IX ou IY.

INC (HL)

Avec INC (HL), la possibilité est donnée d'incrémenter — en ajoutant 1 — le contenu d'un octet de la mémoire. C'est le mode d'adressage indirect qui est utilisé.

Exemple : MODE D'ADRESSAGE INDIRECT (14 octets)

Programme BASIC

```
10
20
30 INPUT "PREMIER NOMBRE " ; N : POKE 43850 , N
40 INPUT "DEUXIEME NOMBRE " ; M : POKE 43851 , M + 1
50 CALL 43801
60 PRINT " LA SOMME VAUT " ; PEEK (43850)
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	21 4A AB		LD	HL,43850
2	3A 4B AB		LD	A,(43851)
3	D6 01	SUITE:	SUB	1
4	28 03		JR	Z,FIN (+3)
5	34		INC	(HL)
6	18 F9		JR	SUITE (−7)
7	C9	FIN:	RET	

La lecture des lignes BASIC indique qu'il s'agit là d'un programme d'addition. L'instruction ADD n'apparaît pas dans la partie assembleur

car elle a été remplacée par une boucle incrémentant le contenu d'un octet et cela le nombre de fois voulu.

Ligne 2 : le contenu de l'octet 43851 est placé dans l'accumulateur, ce qui fait que A contient le nombre M + 1. Cette valeur est égale au second terme de la somme, augmenté de 1.

Ligne 3 : on retranche 1 au registre A. Voici que s'explique la raison pour laquelle on est parti d'une valeur supérieure d'une unité pour A. Ceci compense cela.

Ligne 4 : si la différence porte sur deux chiffres égaux, c'est-à-dire si A vaut 1, un branchement est effectué à la ligne 7, ligne appelée FIN. 28 est le code machine de l'instruction JR Z et 03 est le nombre d'octets que le programme va devoir sauter. Vous rappelez-vous le registre PC, le compteur ordinal ? C'est lui qui va réaliser ce branchement. Il contient toujours l'adresse de la prochaine instruction à exécuter ; donc, dans notre exemple, il est chargé avec le nombre 43811 (faites le compte, le code de l'instruction INC est bien dans cet octet). Ainsi, lorsque le test de la ligne 4 indiquera qu'un branchement est nécessaire, trois unités seront ajoutées au registre PC qui ira pointer sur l'octet 43814, octet correspondant à RET. Retenez de tout cela que le décompte du nombre d'octets dans les opérations de branchement se fait toujours à partir du début de la ligne suivante.

Ligne 5 : on ajoute 1 au contenu de l'octet 43850, donc au premier terme de la somme.

Ligne 6 : JR est l'instruction de branchement inconditionnel. Equivalente de GOTO, elle renvoie le processeur à la ligne 3 pour la suite du programme.

Nous nous retrouvons une nouvelle fois devant une boucle. A chaque passage elle procède aux deux opérations suivantes :

- 1 est enlevé au registre A, c'est-à-dire au deuxième nombre de la somme.
- 1 est ajouté à l'octet 43850, c'est-à-dire au premier terme de la somme.

A étant décrémenté à chaque fois, sa valeur arrivera forcément à 1. La ligne assembleur 3 effectuera alors une différence donnant un résultat nul, et le branchement JR Z sera réalisé. L'octet 43850 con-

tiendra à la fin du programme la somme des deux nombres que nous avons proposés à l'ordinateur.

Reprenons rapidement ce que nous avions déjà dit à propos des additions sur huit bits (voir instruction ADD) : lorsque le résultat d'une somme arrive à 256, il est ramené à zéro.

Exemple :

200 + 100 = 256 + 44 = 44.

DIFFÉRENTES FORMES DE L'INCRÉMENTATION D'UN OCTET MÉMOIRE

INC	(HL)
INC	(IX+n)
INC	(IY+n)
Le contenu de l'octet pointé est incrémenté.	

PUSH — POP

Ces deux instructions permettent l'empilement et le dépilement de registres dans la pile système. Elles n'autorisent que le mode d'adressage registre.

Exemple : MODE D'ADRESSAGE REGISTRE (12 octets)

Programme BASIC

```
 5 MODE 1
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	3E 5A		LD	A,90
2	F5	SUITE:	PUSH	AF
3	CD 5D BB		CALL	47965
4	F1		POP	AF
5	D6 01		SUB	1
6	20 F7		JR	NZ,SUITE (−9)
7	C9		RET	

Lignes 1 et 2 : le programme débute en écrivant dans l'accumulateur le code ASCII de la lettre majuscule Z. Immédiatement après, cette valeur est sauvegardée dans la pile. Peu nous importe de savoir réellement dans quel octet de la mémoire l'empilement a eu lieu. Ce qui nous intéresse est que nous pourrons retrouver la valeur de

l'accumulateur quand le besoin s'en fera sentir. N'accordez qu'une attention distraite au fait que l'instruction PUSH empile non seulement le registre A mais avec lui un certain registre F. Il n'est nullement nécessaire de savoir ce qu'est ce dernier, dans un premier temps du moins. Sachez simplement que PUSH empile toujours deux registres 8 bits et que A ne lui suffit donc pas.

Ligne 3 : nous revoici devant l'appel de la routine d'affichage. Un premier caractère, la lettre Z (ASCII 90) apparaît sur l'écran. En même temps le curseur est déplacé d'une case vers la droite.

Ligne 4 : POP est l'opération inverse de PUSH. Le registre A retrouve alors sa valeur initiale et reprend la valeur 90. Puisque A a la même configuration qu'à la ligne 2, on peut se demander à quoi a servi ce que nous avons fait. A valait 90, il vaut encore 90 ; était-il alors nécessaire de mettre en œuvre la pile ? La réponse est oui, certainement oui, car l'ordinateur utilise l'accumulateur dans l'exécution du sous-programme d'affichage. Et il nous le rend avec une valeur qui n'a rien à voir avec celle que nous lui avions envoyée.

Lignes 5 et 6 : on retranche 1 au registre A qui passe à 89 ; et, puisque ce résultat n'est pas nul, le microprocesseur se rebranche à la ligne SUITE. Ce sera cette fois la lettre Y (ASCII 89) qui va s'écrire sur le téléviseur.

Lorsque le programme s'arrêtera de boucler, 90 caractères seront visibles sur l'écran. Vous remarquerez que l'Amstrad a quelques caractères en réserve, en plus de ceux qui sont standard.

AUTRES FORMES D'UTILISATION DE LA PILE

PUSH	Rd	
POP	Rd	Rd est l'un des registres doubles AF, BC, DE, HL, IX ou IY.

DEC A

Le contenu de l'accumulateur est décrémenté, c'est-à-dire qu'une unité lui est retirée. Le mode d'adressage registre est le seul utilisable.

Exemple : MODE D'ADRESSAGE REGISTRE (22 octets)

Programme BASIC

```
 5 MODE 2
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	3E FA		LD	A,250
2	11 00 00		LD	DE,0
3	21 C8 00		LD	HL,200
4	F5	SUITE:	PUSH	AF
5	D5		PUSH	DE
6	E5		PUSH	HL
7	CD EA BB		CALL	48106
8	E1		POP	HL
9	D1		POP	DE
10	F1		POP	AF
11	13		INC	DE
12	3D		DEC	A
13	20 F3		JR	NZ,SUITE (−13)
14	C9		RET	

Lignes 1, 2 et 3 : on place dans A la valeur 250. Elle va correspondre au nombre de passages dans la boucle SUITE. Les registres DE et HL contiennent les coordonnées d'un point que l'on se propose d'allumer.

Lignes 4, 5 et 6 : on empile les trois registres doubles car leurs contenus vont être ''corrompus'' par la routine appelée à la ligne suivante.

Ligne 7 : voici le deuxième sous-programme de ce livre. Il est tout fait et réalise les fonctions d'allumage d'un point graphique sur le téléviseur. L'ordinateur est lancé dans ce sous-programme et fait pour nous le même travail que celui qui est effectué par l'instruction BASIC PLOT. Comment ? Nous n'en savons rien. Mais une chose est sûre : le point de coordonnées (0, 200) apparaît sur l'écran. Notons, nous devrons nous en resservir, que son abscisse (0) et son ordonnée (200) ont été placées dans les registres DE et HL avant l'appel du sous-programme numéro 48106.

Lignes 8, 9 et 10 : quand le microprocesseur retrouve le cheminement normal de notre programme, les valeurs des registres AF, DE et HL n'ont plus de rapport avec ce qu'elles étaient. On transfère donc les nombres sauvegardés vers les registres qui retrouvent alors leurs valeurs initiales ; sans perdre de vue qu'il est obligatoire que le dépilement et l'empilement soient faits dans un ordre inverse l'un de l'autre.

Lignes 11 et 12 : on augmente d'une unité le contenu de DE et on décrémente l'accumulateur. DE vaut donc à ce moment-là 1 et A, 249. Oui ?

Ligne 13 : la soustraction DEC n'ayant pas fait apparaître un résultat nul, le programme se rebranche à la ligne 4 pour une deuxième exécution de la boucle SUITE.

En plus des nécessaires opérations de pile, l'ordinateur allume un deuxième point, de coordonnées 1 et 200, incrémente DE et décrémente A. Puis se relance dans la boucle pour allumer un troisième point, un quatrième, etc.

Quand le 250e point sera allumé, la soustraction de la ligne 12 donnera un résultat nul et le programme se terminera. Une droite horizontale formée de 250 points pourra se voir sur l'écran.

DIFFÉRENTES FORMES DE LA DÉCRÉMENTATION D'UN REGISTRE

DEC	R	R est l'un des registres A, B, C, D, E, H ou L.
DEC	Rd	Rd est l'un des registres BC, DE, HL, IX ou IY.

DEC (HL)

Cette instruction permet de retirer 1 de la valeur d'un octet de la mémoire. On emploie le mode d'adressage indirect.

Exemple : MODE D'ADRESSAGE INDIRECT (13 octets)

Programme BASIC

```
10
20
30 MODE 0 : LOCATE 5,10 : PRINT " DEBUT "
40 CALL 43801 : LOCATE 5,10 : PRINT " FIN "
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	3E FF		LD	A,255
2	21 4A AB		LD	HL,43850
3	77		LD	(HL),A
4	3D	SUITE:	DEC	A
5	20 FD		JR	NZ,SUITE (−3)
6	35		DEC	(HL)
7	20 FA		JR	NZ,SUITE (−6)
8	C9		RET	

Voici le programme qui réalise une boucle de temporisation. Elle est équivalente en durée de la boucle BASIC :

```
FOR I = 1 TO 250 : NEXT
```

Mais, si le BASIC n'a rien fait d'autre que de compter jusqu'à 250, l'assembleur, pendant le même temps, a eu le loisir d'exécuter sa boucle vide plus de 60 000 fois. Cela met en avant l'extraordinaire rapidité des programmes écrits en langage machine.

Lignes 1, 2 et 3 : le registre A est chargé avec la valeur maximale et cette valeur est écrite dans l'octet 43850.

Lignes 4 et 5 : A valant 255, la décrémentation lui soustrait une unité et le fait passer à 254. On doit être maintenant à l'instruction JR NZ qui va renvoyer le programme à la ligne 4. Ainsi le microprocesseur n'a effectué aucune action visible : il n'a fait que perdre du temps. La ligne 4 place dans A le nombre 253 et, à nouveau, le branchement JR NZ fait retourner à la ligne SUITE. On retrouve donc, avec ces deux lignes, mais en beaucoup plus rapide, la ligne BASIC :

```
FOR I = 254 TO 0 STEP - 1 : NEXT
```

Ligne 6 : l'accumulateur étant alors à zéro, c'est au tour de l'octet 43850 d'être décrémenté. Il avait été chargé au départ avec le nombre 255, il va donc contenir 254.

Ligne 7 : nouvelle utilisation de JR NZ qui concernera la dernière soustraction effectuée, en l'occurrence la décrémentation de la ligne 6. Puisque cette différence aura été faite entre les nombres différents 255 et 1, le programme se rebranchera à SUITE, donc 6 octets en arrière.

On retrouvera alors la ligne 4. A sera une nouvelle fois décrémenté et, partant de 0, repassera à 255. N'oublions pas que – 1 s'écrit pour le processeur 255 (ou FF hexadécimal). Nous voilà de nouveau dans une boucle qui va faire passer A de 255 à 0 (lignes 4 et 5) puis, à terme, une décrémentation de l'octet 43850 (ligne 6) sera réalisée. Puisque cet octet en sera alors à la valeur 253, il y aura encore branchement à la ligne 4.

Le principe doit dès lors être compris : tant que le contenu de l'octet 43850 ne sera pas nul, le programme bouclera. Son exécution aura duré au total moins d'une demi-seconde.

DIFFÉRENTES FORMES DE LA DÉCRÉMENTATION D'UN OCTET MÉMOIRE

DEC	(HL)
DEC	(IX + n)
DEC	(IY + n)
Le contenu de l'octet pointé est décrémenté.	

DJNZ

C'est une instruction de branchement conditionnel : elle décrémente le registre B et ne réalise le branchement que si le contenu du registre n'a pas atteint la valeur 0.

Exemple : MODE D'ADRESSAGE RELATIF (11 octets)

Programme BASIC

```
 5 MODE 1 : LOCATE 1,10
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	21 00 C0		LD	HL,49152
2	06 F0		LD	B,240
3	36 53	SUITE:	LD	(HL),83
4	23		INC	HL
5	10 FB		DJNZ	SUITE (−5)
6	C9		RET	

Lignes 1 et 2 : on fait pointer HL sur le premier octet de la zone mémoire vidéo et on écrit dans B le nombre 240. B nous servira de compteur dans une boucle que l'on va exécuter 240 fois.

Ligne 3 : l'octet 49152 prend la valeur 83. Il va donc colorer le premier segment de l'écran de la manière que nous allons voir.

```
                    7 6 5 4 3 2 1 0  <------ numéros des bits
Octet 49152  -->  |0|1|0|1|0|0|1|1|
```

Nous sommes en mode 1, un segment prend quatre teintes différentes :

Numéro des bits	Valeur binaire de l'encre	Valeur décimale de l'encre	Couleur
3 et 7	00	0	Bleu
2 et 6	01	1	Jaune
1 et 5	10	2	Cyan
0 et 4	11	3	Rouge

En résumé, le premier segment de l'image va apparaître avec les teintes bleue, jaune, cyan et rouge. Cela n'est pas très visible sur notre téléviseur à cause de la haute résolution de l'Amstrad mais rien ne nous empêche d'en demander la confirmation au BASIC :

```
FOR I = 0 TO 7 : PRINT TEST ( I , 399 ) ; : NEXT
```

Réponse : **0 0 1 1 2 2 3 3** ; c'est ce que l'on attendait.

Lignes 4 et 5 : HL est incrémenté et son contenu va alors être relatif au deuxième segment de l'écran. Notre nouvelle instruction est, pour sa part, équivalente de

```
DEC  B
JR   NZ,SUITE
```

Le branchement va donc se réaliser et l'ordinateur affichera, dans les quatre couleurs déjà définies, le deuxième segment vidéo.

Faisons les comptes : le programme sortira de la boucle à la fin des 240 décrémentations de B. On pourra alors voir sur le téléviseur trois droites parallèles. Chacune d'elles est constituée de 80 segments quadrichromes.

Note : Il n'existe pas d'instruction équivalente de DJNZ pour les autres registres.

JR C JR NC

Nous abordons maintenant deux nouvelles instructions de branchement qui vont pouvoir, lorsque les conditions voulues seront réalisées, mettre une nouvelle valeur dans le registre PC et permettre ainsi d'annuler le déroulement séquentiel du programme. Ces instructions vont porter sur la comparaison des grandeurs de deux nombres, dont le premier sera toujours dans un registre. Rappelons que, pour pouvoir utiliser JR Z et JR NZ, il fallait qu'une opération ou une comparaison ait été effectuée auparavant. Il va en être de même pour ces deux nouvelles instructions.

JR NC

Branchement si supérieur ou égal

JR NC réalisera un branchement à l'un des octets du programme machine dans l'un des cas suivants :

1. Soustraction entre deux nombres dont le premier est supérieur ou égal au second.
2. Comparaison entre deux nombres dont le premier, contenu dans l'accumulateur, est supérieur ou égal au second.

L'octet suivant l'instruction JR NC précisera, sur le mode complément à deux, quelle partie du programme devra alors être exécutée. Naturellement, cette instruction sera sans effet si le premier nombre est inférieur à l'autre.

A noter que la comparaison se fera sans que l'ordinateur tienne compte de la valeur en complément à deux de ces nombres. Si, par exemple, les deux nombres valent 254 et 10, le premier sera considéré comme supérieur au second bien que ce soit en réalité le nombre −2 en complément à deux.

Si l'on veut trouver l'équivalent BASIC de JR NC, on doit écrire :

```
IF A >B THEN ...
```

JR C

Branchement si inférieur

Cette fois, le branchement s'effectuera dans l'un des cas suivants :

1. Soustraction entre deux nombres dont le premier est strictement inférieur au second.
2. Comparaison entre deux nombres dont le premier est strictement inférieur au second.

Lors d'une comparaison, le premier nombre est toujours contenu dans l'accumulateur.

Pour cette instruction aussi, l'ordinateur ne tiendra pas compte des valeurs négatives, celles qui correspondent au complément à deux. Écrivons la ligne BASIC correspondante :

```
IF A < = B THEN ...
```

CP

Une comparaison est réalisée entre l'accumulateur et le nombre décrit immédiatement après cette instruction. Une commande de branchement doit suivre normalement cette comparaison. Les modes d'adressage immédiat, registre et indirect peuvent être utilisés.

Exemple : MODE D'ADRESSAGE INDIRECT (25 octets)

Programme BASIC

```
10
20
30 X = INT ( RND * 256 ) : POKE 43850 , X
40 INPUT " QUEL NOMBRE PROPOSEZ-VOUS " ; N :
   POKE 43851 , N
50 CALL 43801 : ON PEEK ( 43852 ) GOTO 60 , 70 , 80
60 PRINT " VOUS AVEZ GAGNE " : END
70 PRINT " TROP GRAND " : GOTO 40
80 PRINT " TROP PETIT " : GOTO 40
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	21 4A AB		LD	HL,43850
2	3A 4B AB		LD	A,(43851)
3	BE		CP	(HL)
4	28 06		JR	Z,EGAL (+6)
5	30 08		JR	NC,SUP (+8)
6	3E 03	INF:	LD	A,3
7	18 06		JR	FIN (+6)
8	3E 01	EGAL:	LD	A,1
9	18 02		JR	FIN (+2)
10	3E 02	SUP:	LD	A,2
11	32 4C AB	FIN:	LD	(43852),A
12	C9		RET	

Voici une version du jeu qui consiste à deviner un nombre que l'ordinateur aura choisi. Le branchement ON GOTO de la ligne BASIC 50 s'effectuera en fonction du nombre trouvé dans l'octet 43852. Voyons comment l'assembleur y aura placé la valeur correcte 1, 2 ou 3.

Ligne 2 : la ligne BASIC 40 aura écrit dans l'octet 43851 le nombre N proposé. C'est donc le registre A qui va contenir ce nombre.

Ligne 3 : une comparaison est effectuée entre le nombre proposé et le contenu de l'octet 43850. Or, dans cet octet a été inscrit par POKE le nombre X que l'ordinateur a tiré au hasard. Voici donc la ligne qui va réaliser la comparaison sur laquelle est basé tout le programme.

Ligne 4 : si la comparaison a porté sur deux nombres égaux, cela voudra dire que l'on a gagné. JR Z va procéder alors à un branchement vers la ligne 8, six octets plus loins. LD A,1 va, à ce moment-là, placer dans l'accumulateur le nombre 1 et un branchement inconditionnel (ligne 9) va entraîner le processeur à l'avant-dernière ligne du programme. Il ne restera plus alors qu'à écrire la valeur 1 dans l'octet 43852. Le BASIC retrouvera ce nombre et l'instruction ON GOTO fera imprimer le message ''VOUS AVEZ GAGNE''.

Ligne 5 : si l'on suppose maintenant que A est supérieur au nombre choisi par l'ordinateur, l'instruction JR NC nous conduira à la ligne 10. Le registre A sera chargé avec la valeur 2, valeur qui sera ensuite écrite (ligne 11) dans l'octet 43852. Il suffira au BASIC de retrouver le contenu de cet octet et le message ''TROP GRAND'' sera affiché sur l'écran.

Lignes 6 et 7 : dernière possibilité enfin, on a proposé à la machine un nombre trop petit. Les instructions JR Z et JR NC sont restées sans effet et le programme s'est déroulé séquentiellement jusqu'à ces lignes. C'est le chiffre 3 qui sera écrit dans l'accumulateur avant qu'un branchement inconditionnel n'envoie le microprocesseur à la ligne 11. 3 est alors recopié dans l'octet 43852 et l'instruction BASIC ON GOTO donnera la réponse à notre essai : ''TROP PETIT''.

DIFFÉRENTES FORMES DE L'INSTRUCTION DE COMPARAISON

CP	n8	Comparaison entre A et une valeur 8 bits.
CP	R	Comparaison entre A et l'un des registres 8 bits R.
CP CP CP	(HL) (IX+n) (IY+n)	Comparaison entre A et le contenu d'un octet mémoire.

OR

Un OU logique est effectué entre l'accumulateur A d'une part et soit un nombre 8 bits, soit un registre simple, soit le contenu d'un octet d'autre part. Les modes d'adressage immédiat, registre et indirect sont donc permis.

Exemple : MODE D'ADRESSAGE IMMÉDIAT (19 octets)

Programme BASIC

```
10
20
30 INPUT " DONNEZ UN NOMBRE " ; N
40 POKE 43850,N : CALL 43801
50 ON PEEK ( 43851 ) GOTO 60 , 70
60 PRINT " LE NOMBRE EST IMPAIR " : GOTO 30
70 PRINT " LE NOMBRE EST PAIR " : GOTO 30
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	21 4A AB		LD	HL,43850
2	7E		LD	A,(HL)
3	F6 01		OR	1
4	BE		CP	(HL)
5	20 04		JR	NZ,PAIR (+4)
6	3E 01		LD	A,1
7	18 02		JR	FIN (+2)
8	3E 02	PAIR:	LD	A,2
9	32 4B AB	FIN:	LD	(43851),A
10	C9		RET	

Réservons quelques lignes pour revoir de quelle façon s'exécute un OU logique entre deux nombres :

```
       1100
    OU 1010
    -------
    =  1110
```

Notre programme, pour sa part, va effectuer un OU entre le contenu du registre A et le nombre 1. Puisque 1 s'écrit en binaire 00000001, seul le dernier bit de A sera concerné. Ainsi, si A se termine par 0, il se verra modifié car son dernier chiffre passera à 1. Par contre, si son dernier chiffre vaut 1, A gardera la même valeur.

Lignes 1 et 2 : le nombre que l'on a proposé à l'ordinateur a été rentré par POKE dans l'octet 43850 et c'est donc le registre A qui est chargé avec cette valeur.

Ligne 3 : le OU logique est réalisé entre le nombre que nous avons choisi et l'unité. Si ce nombre est pair, son écriture binaire se fera avec un 0 à la fin et, s'il est impair, il se terminera par 1. L'action de OR va donc consister à modifier la valeur de notre registre uniquement dans le cas où il est pair.

Lignes 4 et 5 : comparaison est faite entre les contenus de l'accumulateur et de l'octet 43850, octet qui, ne l'oublions pas, contient le nombre que nous avons tapé au clavier. Dans le cas où ce nombre est pair, OR l'a transformé et un branchement à la ligne 8 est effectué.

Lignes 6 et 7 : s'il s'agit d'un nombre impair, la valeur 1 est mise dans A pour être récrite ensuite (ligne 9) dans l'octet 43851.

Ligne 8 : sinon, c'est le nombre 2 qui va alors transiter par l'accumulateur pour être placé ensuite dans ce même octet.

La ligne BASIC 50 va alors examiner cet octet et le branchement ON GOTO enverra à ce moment l'ordinateur à la bonne instruction.

DIFFÉRENTES FORMES DU **OU** LOGIQUE

OR	n8	n8 est un nombre 8 bits.			
OR	R	R est l'un des registres simples.			
OR	(HL)	OR	(IX+n)	OR	(IY+n)

AND

Un ET logique est réalisé entre l'accumulateur et une valeur 8 bits prise comme une donnée, un contenu registre ou un contenu mémoire. On peut utiliser les modes d'adressage immédiat, registre ou indirect.

Exemple : MODE D'ADRESSAGE IMMÉDIAT (15 octets)

Programme BASIC

```
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	3E 5A		LD	A,90
2	F5	SUITE:	PUSH	AF
3	CD 5D BB		CALL	47965
4	F1		POP	AF
5	3D		DEC	A
6	E6 FE		AND	254
7	FE 41		CP	65
8	30 F4		JR	NC,SUITE (−12)
9	C9		RET	

Ligne 2 : on sauvegarde dans la pile la valeur que l'on vient d'écrire dans A, la routine 47965 qui va être appelée utilisant pour son propre compte l'accumulateur.

Lignes 3 et 4 : la lettre Z, dont le code ASCII est 90, est affichée sur l'écran. Puis le registre A reprend sa valeur de départ.

Lignes 5 et 6 : l'instruction de décrémentation remplace 90 par 89 et l'ordinateur exécute notre nouvelle instruction en réalisant un ET logique entre les nombres 89 et 254. Voyons cela au niveau du binaire :

```
   0 1 0 1 1 0 0 1 <———— 89
ET 1 1 1 1 1 1 1 0 <———— 254
———————————————————
=  0 1 0 1 1 0 0 0 <———— 88
```

Nous retrouvons la particularité déjà mentionnée de la commande AND. Elle permet de forcer à 0 n'importe quel chiffre binaire d'un nombre. En ce qui nous concerne, c'est le dernier d'entre eux qui est passé à 0.

Ligne 7 : l'accumulateur possédant une valeur supérieure à 65, le programme retrouve la ligne SUITE. Une deuxième lettre est alors imprimée sur le téléviseur, juste à côté de la précédente. Son code ASCII est égal à 88 ; c'est donc de la lettre X qu'il s'agit.

Continuons à suivre le travail du microprocesseur :

DEC A : le registre A passe à 87 (soit 01010111 binaire)
AND 254 : ce même registre perd son dernier chiffre 1 et vaut alors 86 (01010110 binaire)

Quand le sous-programme d'affichage sera appelé, c'est la lettre V qui va cette fois apparaître. Inutile de poursuivre plus avant. Vous avez compris que ce programme écrit l'alphabet en ordre décroissant et en sautant une lettre sur deux. La dernière d'entre elles sera la lettre B (ASCII 66).

DIFFÉRENTES FORMES DU ET LOGIQUE

AND	n8	n8 est un nombre 8 bits.			
AND	R	R est l'un des registres simples.			
AND	(HL)	AND	(IX+n)	AND	(IY+n)

XOR

Comme les deux instructions précédemment étudiées, XOR réalise une opération logique : un OU exclusif est effectué entre le registre A et un nombre de 8 bits. On peut là aussi utiliser les modes d'adressage immédiat, registre et indirect.

Exemple : MODE D'ADRESSAGE IMMÉDIAT (14 octets)

Programme BASIC

```
 5 MODE 2
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	06 50		LD	B,80
2	21 C0 C3		LD	HL,50112
3	3E F7		LD	A,247
4	77	SUITE:	LD	(HL),A
5	EE FF		XOR	255
6	23		INC	HL
7	10 FA		DJNZ	SUITE (−6)
8	C9		RET	

Lignes 1 et 2 : les registres B et HL sont chargés respectivement avec les valeurs 80 et 50112. Le premier registre va nous servir de compteur et le second pointera sur 80 octets consécutifs de la mémoire vidéo.

Lignes 3 et 4 : la valeur de l'accumulateur est recopiée dans l'octet d'adresse 50112. Cet octet correspond à un segment situé à gauche de l'écran et approximativement en son milieu (50112 = 49152 + 12 * 80). Le mode 2 ayant été choisi, le segment a la possibilité de voir ses huit points allumés indépendamment les uns des autres. En regardant la décomposition binaire de 247 (11110111) on déduit que seul le quatrième point en partant de la droite ne sera pas visible.

Ligne 5 : l'utilisation du OU exclusif est faite ici dans le but d'inverser tous les chiffres d'un nombre binaire. En effet, quand XOR est effectué entre un chiffre et 1, ce chiffre change de valeur.

Pour notre exemple, il vient :

```
      1 1 1 1 0 1 1 1 <-------- 247 (registre A)
  XOR 1 1 1 1 1 1 1 1 <-------- 255
  -------------------
  =   0 0 0 0 1 0 0 0 <-------- 8   (registre A)
```

Ligne 7 : le contenu de B est abaissé d'une unité et prend la valeur 79. Le programme se rebranche donc à la ligne 4. Auparavant, HL s'est vu incrémenté et pointe de ce fait sur l'octet qui s'occupe du segment placé à droite de celui qui est déjà visible. Par écriture de la valeur 8 dans l'octet 50113, on allume le quatrième point en partant de la droite de ce deuxième segment.

Il n'y a plus qu'à analyser ce que donne une nouvelle utilisation de XOR.

```
      0 0 0 0 1 0 0 0 <-------- 8   (registre A)
  XOR 1 1 1 1 1 1 1 1 <-------- 255
  -------------------
  =   1 1 1 1 0 1 1 1 <-------- 247 (registre A)
```

A retrouve alors sa valeur d'origine. Voilà pourquoi notre programme fait apparaître en fin d'exécution 80 segments, chacun d'entre eux étant le négatif (au sens photographique) du précédent.

DIFFÉRENTES FORMES DU **OU** EXCLUSIF

XOR	n8	n8 est un nombre 8 bits.			
XOR	R	R est l'un des registres simples.			
XOR	(HL)	XOR	(IX+n)	XOR	(IY+n)

SRL A

Abréviation de Shift Right Logical, *cette instruction décale tous les bits de l'accumulateur A vers la droite. Le bit de gauche est mis à zéro. Il s'agit du mode d'adressage registre.*

Exemple : MODE D'ADRESSAGE REGISTRE (9 octets)

Programme BASIC

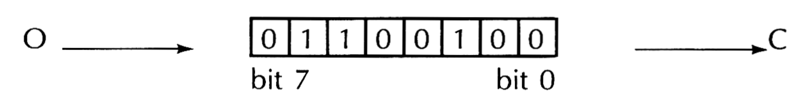
```
10
20
30 PRINT " DONNEZ UN NOMBRE PLUS PETIT QUE 256 " ;
40 INPUT N : POKE 43850 , N : CALL 43801
50 PRINT " LE QUOTIENT ENTIER DU NOMBRE " ;
60 PRINT " PAR DEUX VAUT " ; PEEK (43851) : GOTO 30
```

Programme assembleur

Lignes	Codes machine	Assembleur
1	3A 4A AB	LD A,(43850)
2	CB 3F	SRL A
3	32 4B AB	LD (43851),A
4	C9	RET

Ce programme effectue en assembleur la division entière d'un nombre par deux. Voyons, au niveau du binaire, comment cela se passe.
Considérons le nombre décimal 100 qui s'écrit en binaire 01100100.





Faisons subir aux chiffres qui constituent ce nombre un décalage vers la droite. Chaque chiffre va se retrouver dans le bit de rang immédiatement inférieur : le chiffre du bit 7 (0) va passer dans le bit 6 ; le chiffre du bit 6 va s'écrire dans le bit 5, etc. Le dernier chiffre à droite (bit 0) va donc sortir de l'octet et sera perdu pour nous. L'ordinateur, lui, en gardera la trace en le mettant dans un endroit spécial qu'on appelle l'indicateur de retenue et que l'on note C. Cet indicateur prendra donc la valeur 0, mais, répétons-le, cela n'a aucune espèce d'importance pour notre exemple.

Sachant que SRL remplace toujours le bit 7 par 0, voici ce que l'on obtient alors pour le registre A :

0	0	1	1	0	0	1	0

La traduction en décimal de cette valeur donne 50 ; on a donc bien divisé le nombre 100 par 2.

Et si nous étions partis d'un nombre impair ? Essayons avec 101.

O ⟶ | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | ⟶ C

Quand SRL aura agi, on obtiendra :

0	0	1	1	0	0	1	0

c'est-à-dire 50, ce qui est bien le quotient entier de 101 par 2. Dans ce deuxième cas, l'indicateur de retenue est passé à 1.

Il ne reste plus qu'à comprendre pourquoi le décalage à droite des chiffres a conduit à une division par deux. Prenons par exemple le chiffre 1 du bit 6 et voyons ce qu'il devient : il valait 2^6, c'est-à-dire 64, avant SRL, il vaut 2^5, soit 32, après ; il a donc été réduit de moitié. Ce même raisonnement se fait pour tous les autres chiffres, ce qui nous donne l'explication.

Les différentes lignes ne seront pas étudiées une à une cette fois, le programme assembleur se comprenant sans difficulté.

DIFFÉRENTES FORMES DE LA ROTATION LOGIQUE VERS LA DROITE (MODE REGISTRE)

SRL	R	R est l'un des registres 8 bits.

SRL (HL)

Comme pour SRL A, c'est un décalage vers la droite, mais ce décalage concernera le contenu d'un octet mémoire. Le mode d'adressage permis est donc l'indirect.

Exemple : MODE D'ADRESSAGE INDIRECT (22 octets)

Programme BASIC

```
 5 MODE 2 : LOCATE 1,10
10
20
30 CALL 43801
40 FOR I = 1 TO 7 : FOR J = 1 TO 200 : NEXT : CALL 43812 : NEXT
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	21 00 C0		LD	HL,49152
2	06 F0		LD	B,240
3	36 80	SUITE1:	LD	(HL),128
4	23		INC	HL
5	10 FB		DJNZ	SUITE1 (−5)
6	C9		RET	
7	21 00 C0		LD	HL,49152
8	06 F0		LD	B,240
9	CB 3E	SUITE2:	SRL	(HL)
10	23		INC	HL
11	10 FB		DJNZ	SUITE2 (−5)
12	C9		RET	

Lignes 1 et 2 : c'est l'initialisation du programme. On partira du premier segment de l'écran et B décomptera le nombre de passages dans la boucle SUITE1.

Lignes 3 à 6 : 128 (10000000 binaire) est écrit dans l'octet 49152, ce qui allume le premier point du segment correspondant. On ajoute alors 1 à HL qui va pointer sur le segment situé exactement à côté du précédent. Lui aussi verra son premier point allumé au second passage de la boucle. Quand les 240 décrémentations du registre B auront été réalisées, nous aurons devant les yeux autant de points allumés. Ils seront répartis à intervalles réguliers (de la largeur d'un caractère) sur trois lignes du téléviseur. Le retour au BASIC sera alors programmé et celui-ci devra alors exécuter la boucle FOR NEXT I.

Cette boucle laisse passer un peu de temps et renvoie le microprocesseur dans le programme machine. Mais attention, cette fois l'exécution a lieu à partir de l'octet 43812, c'est-à-dire à la ligne 7 de l'assembleur.

Lignes 7, 8 et 9 : on repart du début de la mémoire écran et l'octet 49152 subit un décalage vers la droite. Puisqu'il valait 10000000 en binaire, il vaudra, toujours en binaire, 01000000 et c'est donc le deuxième point du premier segment de l'image qui va s'allumer.

Lignes 10 et 11 : on charge le registre HL avec l'adresse du segment placé à côté du précédent. Et l'on retrouve une boucle qui colorera le deuxième point de chacun des 240 premiers segments de l'écran, le premier point s'étant effacé.

On ressort à ce moment-là du programme assembleur, on effectue une temporisation grâce à l'ensemble FOR NEXT J et on se replace à la ligne 7 pour y éclairer cette fois le troisième point de nos 240 segments. La boucle BASIC est exécutée sept fois, ce qui fait qu'au total les huit points de nos segments auront été, les uns à la suite des autres, rendus visibles.

DIFFÉRENTES FORMES DE LA ROTATION LOGIQUE VERS LA DROITE (MODE INDIRECT)

SRL (HL)	SRL (IX+n)	SRL (IY+n)

SLA A

C'est cette fois d'un décalage vers la gauche qu'il est question, SLA étant l'abréviation de Shift Left Arithmetic. *Le bit 7 passe dans l'indicateur de retenue et le bit 0 du registre A est mis à zéro. SLA est utilisée ici avec le mode d'adressage registre.*

Exemple : MODE D'ADRESSAGE REGISTRE (9 octets)

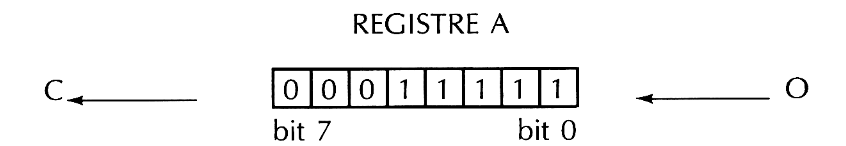
Programme BASIC

```
10
20
30 INPUT " DONNEZ UN NOMBRE " ; N : POKE 43850 , N
40 J = 1 : FOR I = 1 TO 3
50 J = J * 2 : CALL 43801
60 PRINT " LE PRODUIT DU NOMBRE PAR " ; J ;
70 PRINT " VAUT " ; PEEK ( 43850 )
80 NEXT : GOTO 30
```

Programme assembleur

Lignes	Codes machine	Assembleur
1	3A 4A AB	LD A,(43850)
2	CB 27	SLA A
3	32 4A AB	LD (43850),A
4	C9	RET

On suppose que l'on propose à l'ordinateur le nombre 31 et on regarde ce qu'il devient quand on exécute l'instruction SLA. 31 a pour équivalent binaire 00011111.



Le décalage à gauche va faire sortir du registre le contenu du bit 7 qui ira se placer dans l'indicateur de retenue, indicateur dont l'importance est nulle en l'état d'avancement de nos connaissances. Tous les chiffres étant translatés, on obtient pour A :

0	0	1	1	1	1	1	0

Le nombre 0 est venu prendre la place laissée libre dans le bit 0. En transcrivant le résultat obtenu en décimal, on obtient le nombre 62, c'est-à-dire le double de 31. Ainsi le décalage à gauche de tous les bits a permis d'effectuer le produit par 2 du nombre qui se trouvait dans l'accumulateur. Cela peut se comprendre puisque, en définitive, chaque chiffre se retrouvera avec une puissance de 2 supérieure d'une unité à la précédente.

Revenons à notre programme : le nombre que l'on a donné au départ à l'ordinateur est placé dans l'octet 43850 et, au premier passage de la boucle FOR NEXT, ce nombre est multiplié par 2. La ligne assembleur 3 replace la réponse dans ce même octet 43850. Au deuxième passage, le nombre est à nouveau multiplié par 2, ce qui fait que la valeur de début est, cette fois, multipliée par 4 ; elle le sera par 8 quand le programme BASIC sera terminé.

Bien entendu, ce programme donne des réponses cohérentes tant que l'on ne choisit pas un nombre supérieur ou égal à 32 (soit 00100000 en binaire). A partir de cette valeur, en effet, c'est un chiffre 1 qui est perdu dans les décalages rendant le résultat final incorrect (mais explicable).

DIFFÉRENTES FORMES DE L'INSTRUCTION SLA

SLA	R	R est l'un des registres 8 bits.

SLA (HL)

Un décalage d'un bit vers la gauche du contenu d'un octet mémoire est effectué. Le bit 0 passe à 0 et le bit 7 est écrit dans l'indicateur de retenue.

Exemple : MODE D'ADRESSAGE INDIRECT (27 octets)

Programme BASIC

```
 5 MODE 2
10
20
30 FOR I = 49231 TO 49152 STEP - 1
40 POKE 43851 , INT ( I / 256 )
50 POKE 43850 , I - INT ( I / 256 ) * 256
60 CALL 43801 : NEXT
```

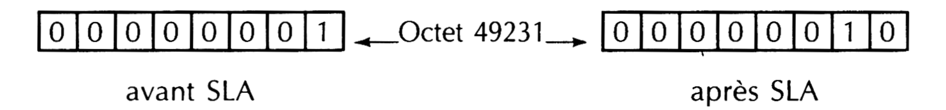
Programme assembleur

Lignes	Codes machine	Assembleur		
1	2A 4A AB		LD	HL,(43850)
2	36 01		LD	(HL),1
3	16 08		LD	D,8
4	CD 29 AB	SEGMENT:	CALL	TEMPO
5	CB 26		SLA	(HL)
6	15		DEC	D
7	20 F8		JR	NZ,SEGMENT(−8)
8	C9		RET	
9	06 32	TEMPO:	LD	B,50
10	C5	SUITE1:	PUSH	BC
11	06 64		LD	B,100
12	10 FE	SUITE2:	DJNZ	SUITE2(−2)
13	C1		POP	BC
14	10 F8		DJNZ	SUITE1(−8)
15	C9		RET	

Lorsque le BASIC appelle pour la première fois le programme assembleur, le nombre 49231 est écrit à cheval sur les octets 43850 et 43851 ; comme toujours le poids faible (79) dans le premier et le poids fort (192) dans l'autre.

Lignes 1, 2 et 3 : le contenu de HL est porté à 49231 et l'octet pointé par ce registre prend la valeur 1. Cet octet vidéo correspond à un segment placé en haut et à droite de l'écran, segment qui nous apparaît donc avec son dernier point allumé.

Lignes 4 à 8 : afin que l'on puisse y voir quelque chose, on oblige le microprocesseur à compter jusqu'à 5000 en l'envoyant exécuter le sous-programme TEMPO. Puis on décale vers la gauche tous les bits de l'octet 49231 :



avant SLA　　　　　　après SLA

Cette fois, c'est le deuxième point en partant de la droite qui va s'éclairer. On renouvelle ce même type d'opération sur la structure de l'octet 49231 tant que le registre D n'a pas atteint 0. Lorsque cela sera fait, tous les points du segment auront successivement été rendus visibles puis éteints.

Quand l'instruction RET de la ligne 8 redonne le contrôle au BASIC, le segment 49231 n'a plus aucun de ses points allumé. La boucle FOR NEXT va alors s'intéresser à l'octet 49230 et transmettre cette valeur au registre HL par l'intermédiaire des octets 43850 et 43851. Le segment 49230 subira les mêmes transformations que le segment 49231 : il verra les huit points qui le constituent s'éclairer puis s'éteindre. Cette action sera ensuite réalisée sur chacun des 78 segments qui forment la première ligne de l'écran. Au total, nous aurons l'impression de voir un point traverser le haut de l'image de la droite vers la gauche.

DIFFÉRENTES FORMES DE L'INSTRUCTION SLA (MODE INDIRECT)

SLA (HL)	SLA (IX+n)	SLA (IY+n)

RL A

Tous les bits de l'accumulateur subissent une rotation vers la gauche. Le bit 7 passe dans l'indicateur de retenue et la valeur préalablement contenue par celui-ci est transférée dans le bit 0. RL A est l'abréviation de Rotate Left A.

Exemple : MODE D'ADRESSAGE REGISTRE (24 octets)

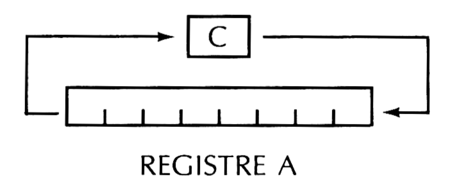
Programme BASIC

```
10
20
30 INPUT " DONNEZ UN NOMBRE " ; N : POKE 43850 , N
40 CALL 43801 : PRINT " SON DOUBLE VAUT " ;
50 PRINT 256 * PEEK (43851) + PEEK (43852) : GOTO 30
```

Programme assembleur

Lignes	Codes machine	Assembleur	
1	DD 21 4A AB	LD	IX,43850
2	3E 00	LD	A,0
3	06 00	LD	B,0
4	CB 17	RL	A
5	DD 7E 00	LD	A,(IX+0)
6	CB 17	RL	A
7	CB 10	RL	B
8	DD 70 01	LD	(IX+1),B
9	DD 77 02	LD	(IX+2),A
10	C9	RET	

Vous vous souvenez de l'instruction SLA ? Elle nous avait permis de multiplier un nombre par 2, 4 ou 8 mais cela n'était pas allé sans un ennui de taille : les chiffres 1 qui sortaient sur la gauche de l'accumulateur étaient perdus et, si l'on partait d'un nombre trop grand, la réponse n'était pas celle attendue. Regardons comment nous allons pouvoir y remédier avec notre instruction RL A :

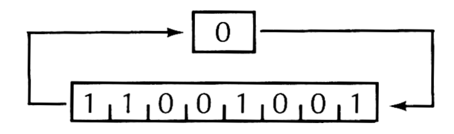


Tous les bits de l'accumulateur subissent un décalage vers la gauche ; le bit contenu dans l'indicateur de retenue passe dans le bit 0 et c'est le bit 7 qui prend sa place. Il s'agit donc là d'une rotation réalisée sur 9 bits.

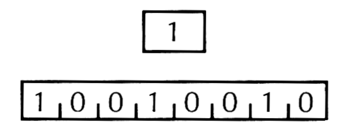
Lignes 2 et 3 : les deux registres 8 bits sont mis à zéro.

Ligne 4 : on fait subir à A une rotation ; puisque A s'écrit 00000000 en binaire, cela n'a pas d'autre effet que de faire rentrer le chiffre 0 dans l'indicateur de retenue.

Lignes 5 et 6 : on recopie dans A le nombre que nous avons écrit par POKE dans l'octet 43850 et, grâce à RL A, on le multiplie par 2. Examinons cela de plus près et supposons, pour fixer les idées, que N ait été choisi égal à 201 (soit 11001001).



C est à zéro (ligne 4) et on obtient donc après RL A :



Le bit C est passé à 1 et la nouvelle valeur de l'accumulateur est, en décimal, 146. Cela n'est naturellement pas le double de 201, mais attendons la suite.

Lignes 7 et 8 : RL B a pour effet de décaler les 8 chiffres 0 du registre B et de faire entrer sur sa droite le bit qui se trouvait dans l'indicateur, c'est-à-dire le bit 1. La nouvelle valeur de B est donc 1 ; elle est inscrite alors dans l'octet 43851.

Lignes 9 et 10 : le décimal 146 est, pour sa part, placé à l'adresse 43852 et le retour au BASIC est programmé.

On va pouvoir vérifier la logique du programme assembleur :

```
PRINT 256 * PEEK ( 43851 ) + PEEK ( 43852 )
```

Réponse : **256 * 1 + 146 = 402**

Tout s'est donc passé en définitive comme si nous avions fait un décalage sur 9 bits:

011001001 (201 décimal) serait devenu
110010010 (402 décimal)

DIFFÉRENTES FORMES DE LA ROTATION VERS LA GAUCHE (MODE REGISTRE)

RL	R	R est l'un des registres 8 bits.

RL (HL)

Tous les bits de l'octet mémoire spécifié sont décalés d'une position vers la gauche. Le bit 7 est placé dans l'indicateur de retenue et la valeur d'origine de celui-ci est transférée dans le bit 0.

Exemple : MODE D'ADRESSAGE INDIRECT (19 octets)

Programme BASIC

```
 5 MODE 2
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	21 00 C0		LD	HL,49152
2	11 D0 07		LD	DE,2000
3	3E 80	SUITE:	LD	A,128
4	CB 17		RL	A
5	CB 16		RL	(HL)
6	23		INC	HL
7	1B		DEC	DE
8	7A		LD	A,D
9	B3		OR	E
10	20 F4		JR	NZ,SUITE(−12)
11	C9		RET	

Lignes 1 et 2 : le registre HL pointe en début de programme sur le premier segment de l'écran. DE, quant à lui, servira de compteur dans une boucle qui va être exécutée 2000 fois. Les 2000 segments

qui seront concernés sont répartis de la façon suivante sur l'image : les 80 premiers sont ceux de la première ligne, les 80 suivants sont ceux de la neuvième ligne ... les 80 derniers sont ceux de la 193e ligne. Ils ont des adresses consécutives prises dans l'intervalle 49152 – 51151.

Lignes 3 et 4 : l'accumulateur se voit attribuer la valeur 128, soit 10000000 binaire, et une rotation vers sa gauche est réalisée. Comme on ne sait pas ce que contient l'indicateur de retenue, on ne sait pas non plus quelle est à ce moment précis la valeur exacte du registre A. La seule chose dont on soit certain est que le bit C est maintenant passé à 1.

Ligne 5 : on fait subir à l'octet 49152 une rotation. Considérant d'une part que cet octet avait la valeur nulle après l'exécution de la commande BASIC MODE 2 et d'autre part que l'indicateur de retenue vaut 1, on déduit la nouvelle valeur de l'octet 49152 : c'est 1. Le point de droite du premier segment de l'écran s'allume donc en jaune.

Ligne 6 : HL, incrémenté, pointe alors sur le deuxième segment du téléviseur, segment qui, lui aussi, va voir son dernier point allumé. Puis cette action se reproduira près de 2000 fois, faisant apparaître au total 2000 points sur l'écran.

Il n'y a aucune difficulté à comprendre la logique de ce programme. Attardons-nous quand même sur la méthode qui a été utilisée pour faire sortir le microprocesseur de la boucle SUITE après les 2000 exécutions souhaitées. Les lignes 8 et 9 opèrent un OU logique entre les registres D et E (par l'intermédiaire de l'accumulateur). Tant que le résultat de cette opération est non nul, le programme se rebranche à la ligne 3. La seule possibilité de retourner au BASIC est d'avoir la valeur nulle dans D et en même temps dans E, c'est-à-dire dans DE. Cette méthode, pour artificielle qu'elle paraisse, est très souvent employée lorsque l'on décrémente un registre double car l'instruction JR ne fonctionne que dans cette situation. Il vaut mieux le savoir, non ?

DIFFÉRENTES FORMES DE LA ROTATION SUR LA GAUCHE (MODE INDIRECT)

RL (HL)	RL (IX+n)	RL (IY+n)

RR A

Cette instruction effectue une rotation vers la droite de tous les bits de l'accumulateur. Le bit de retenue prend la place du bit 7 ; il est lui-même remplacé par le bit 0.

Exemple : MODE D'ADRESSAGE REGISTRE (27 octets)

Programme BASIC

```
 5 MODE 0
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine		Assembleur	
1	06 0A		LD	B,10
2	3E 62		LD	A,98
3	CD 2A AB	SUITE:	CALL	AFFICH
4	CB 1F		RR	A
5	CD 2A AB		CALL	AFFICH
6	CB 17		RL	A
7	10 F4		DJNZ	SUITE (−12)
8	C9		RET	
9	F5	AFFICH:	PUSH	AF
10	C5		PUSH	BC
11	CD 5D BB		CALL	47965
12	C1		POP	BC
13	F1		POP	AF
14	37		SCF	
15	3F		CCF	
16	C9		RET	

Lignes 1 et 2 : les registres B et A sont initialisés avec les valeurs 10 et 98. Le premier tiendra le rôle de compteur dans la boucle SUITE ; le second contient pour l'instant le code ASCII de la lettre minuscule b.

Ligne 3 : l'ordinateur part exécuter le programme AFFICH. Il commence par sauvegarder les registres AF et BC puis appelle la routine 47965. Nous voici en terrain connu puisque nous savons que ce sous-programme réalise l'affichage du caractère dont le code est contenu par l'accumulateur. En haut et à gauche de l'écran apparaît donc la lettre b. On ressort de la pile les registres BC et AF et cela a pour effet de redonner aux registres B et A (les seuls qui nous intéressent dans cet exemple) les valeurs 10 et 98. Puis on oblige le bit de retenue, le bit C, à s'annuler. Cela est cette fois réalisé grâce à la succession des instructions SCF et CCF. La première force le bit C à prendre la valeur 1 et la seconde le force à prendre la valeur binaire opposée (c'est-à-dire 0 dans ce cas). Il n'existe pas d'instruction assembleur qui suffise à elle seule à annuler le bit de retenue.

Ligne 4 : on effectue une rotation vers la droite de l'accumulateur à travers l'indicateur de retenue.

A valait 98 décimal soit 01100010 binaire et C valait 0
A vaut 49 décimal soit 00110001 binaire et C vaut 0

Ligne 5 : on retrouve le sous-programme AFFICH et c'est cette fois le chiffre 1 (code ASCII 49) qui va s'écrire sur l'écran.

Ligne 6 : nouvelle rotation, mais vers la gauche maintenant ; A reprend alors sa valeur d'origine, 98.

Ligne 7 : le processeur restera dans la boucle SUITE jusqu'à ce que le registre B s'annule. On verra à ce moment-là se succéder sur la première ligne de l'écran 20 caractères, la lettre b et le chiffre 1 apparaissant tour à tour.

DIFFÉRENTES FORMES DE LA ROTATION VERS LA DROITE (MODE REGISTRE)

RR	R	R est l'un des registres 8 bits.

RR (IX+n)

Une rotation vers la droite du contenu d'un octet mémoire est réalisée. Le bit 0 prend la place du bit de retenue qui, lui-même, se retrouve à l'emplacement du bit 7.

Exemple : MODE D'ADRESSAGE INDIRECT (33 octets)

Programme BASIC

```
 5 MODE 2
10
20
30 CALL 43801
```

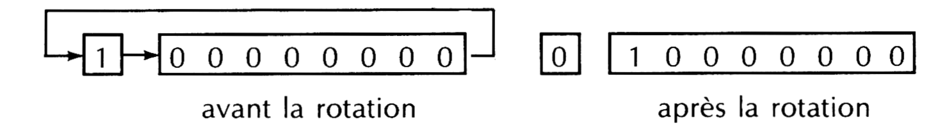
Programme assembleur

Lignes	Codes machine	Assembleur		
1	DD 21 00 C0		LD	IX,49152
2	11 00 08		LD	DE,2048
3	21 D0 07		LD	HL,2000
4	DD E5	DEBUT:	PUSH	IX
5	06 08		LD	B,8
6	37	SUITE:	SCF	
7	DD CB 00 1E		RR	(IX+0)
8	DD 19		ADD	IX,DE
9	10 F7		DJNZ	SUITE (−9)
10	DD E1		POP	IX
11	DD 23		INC	IX
12	2B		DEC	HL
13	7C		LD	A,H
14	B5		OR	L
15	20 EA		JR	NZ,DEBUT(−22)
16	C9		RET	

Lignes 1, 2 et 3 : 49152 est l'adresse du premier octet de la mémoire écran, 2048 est l'écart qu'il y a entre deux segments vidéo superposés, et 2000 est le nombre total de caractères qui peuvent apparaître à l'image en mode 2.

Lignes 4 et 5 : on met de côté la valeur 49152 et on charge le registre B avec le nombre de passages que le programme va effectuer dans la boucle SUITE.

Lignes 6 à 9 : après avoir porté à 1 le contenu du bit de retenue, on effectue une rotation sur la droite de l'octet 49152 :



avant la rotation — après la rotation

Ainsi donc, notre octet passe de la valeur 0 à la valeur 128. Du coup, son point de gauche s'allume. On ajoute alors 2048 à 49152 et l'on retourne à la ligne SUITE. Au sortir de la boucle, 8 points seront visibles sur l'écran, disposés les uns en dessous des autres.

Lignes 10 à 15 : on retrouve la valeur 49152 et on lui ajoute 1. IX pointe donc maintenant sur le deuxième segment de l'image. L'ordinateur renvoie le programme à la quatrième ligne et la boucle SUITE, une nouvelle fois mise à contribution, allume les 8 points de gauche de la deuxième case caractère de l'écran. Puis ce sera le tour de la troisième, de la quatrième, de la deux millième case. Le "décrochage" de l'assembleur aura lieu à ce moment-là puisque les registres H et L seront tous deux nuls. 80 lignes jaunes verticales se dessineront alors sur le téléviseur.

DIFFÉRENTES FORMES DE LA ROTATION VERS LA DROITE (MODE INDIRECT)

RR (HL)	RR (IX+n)	RR (IY+n)

ADC

Cette instruction est l'abréviation de ADd with Carry. *Elle s'utilise comme l'instruction ADD mais la valeur de l'indicateur de retenue C est ajoutée au résultat de l'addition. On peut employer les modes d'adressage immédiat, registre et indirect.*

Exemple : MODE D'ADRESSAGE IMMÉDIAT (19 octets)

Programme BASIC

```
 10
 20
 30 INPUT " PREMIER NOMBRE " ; N1
 40 POKE 43851 , INT (N1 / 256 )
 50 POKE 43850 , N1 - INT ( N1 / 256) * 256
 60 INPUT " DEUXIEME NOMBRE " ; N2
 70 POKE 43853 , INT ( N2 / 256 )
 80 POKE 43852 , N2 - INT ( N2 / 256 ) * 256
 90 CALL 43801 : PRINT " REPONSE " ;
100 PRINT 65536*PEEK(43862)+256*PEEK(43861)+PEEK(43860)
```

Programme assembleur

Lignes	Codes machine	Assembleur	
1	ED 5B 4A AB	LD	DE,(43850)
2	2A 4C AB	LD	HL,(43852)
3	19	ADD	HL,DE
4	22 54 AB	LD	(43860),HL
5	3E 00	LD	A,0
6	CE 00	ADC	A,0
7	32 56 AB	LD	(43862),A
8	C9	RET	

Lignes 1 à 4 : le registre 16 bits DE est chargé avec le nombre N1 ; il lui est ajouté le nombre N2, et le résultat est rangé, sous la forme poids faible/poids fort, dans les octets 43860 et 43861. Le programme pourrait s'arrêter là si nous nous contentions d'ajouter deux nombres ayant une somme plus petite que 65536. Supposons qu'il n'en soit rien et proposons à l'ordinateur le calcul 50000+20000 : il va considérer que 70000 se décompose en 65536 d'une part et en 4464 d'autre part. Cette dernière valeur sera écrite dans les octets 43860 et 43861 mais il va garder la trace du débordement de la capacité 16 bits en forçant à 1 le bit de retenue. Il nous faut voir comment nous allons pouvoir nous servir de cette indication.

Lignes 5 et 6 : ces deux lignes ont pour but d'écrire dans le registre A le chiffre du bit de retenue. On met l'accumulateur à 0 et on lui ajoute alors la retenue et la valeur 0. Au total, A contiendra bien la valeur d'origine de l'indicateur.

Ligne 7 : il ne reste qu'à ranger ce résultat dans l'octet 43862, là où le programme appelant pourra le retrouver.

En définitive, si le calcul de la somme dépasse 16 bits, le nombre 65536 est ajouté au résultat final par la ligne BASIC 100.

DIFFÉRENTES FORMES DE L'ADDITION AVEC RETENUE

<table>
<tr><td>ADC A,n8</td><td colspan="2">n8 est un nombre 8 bits.</td></tr>
<tr><td>ADC A,R</td><td colspan="2">R est l'un des registres 8 bits.</td></tr>
<tr><td>ADC A,(HL)</td><td>ADC A,(IX+n)</td><td>ADC A,(IY+n)</td></tr>
<tr><td>ADC HL,Rd</td><td colspan="2">Rd est l'un des registres doubles BC, DE, HL ou SP.</td></tr>
</table>

LDI

Le contenu de l'octet pointé par HL est recopié dans l'octet pointé par DE. Puis ces deux registres sont incrémentés.

Exemple : MODE D'ADRESSAGE INHÉRENT (11 octets)

Programme BASIC

```
 5 MODE 0 : POKE 49152 , 63 : POKE 49153 , 255
10
20
30 CALL 43801 : LOCATE 1,10
```

Programme assembleur

Lignes	Codes machine	Assembleur	
1	21 00 C0	LD	HL,49152
2	11 4E C0	LD	DE,49230
3	ED A0	LDI	
4	ED A0	LDI	
5	C9	RET	

Ligne BASIC 5 : en mode 0 chaque segment ne peut être coloré qu'avec deux couleurs.

POKE 49152 , 63

Les deux parties du premier segment de l'écran nous apparaissent avec la couleur d'encre 14. Ce segment se met donc à clignoter en

bleu sur jaune. En effet 63 s'écrit 00111111 en binaire, ce qui correspond à la répartition suivante :

1 5 3 7 ← numéros des bits →	0 4 2 6
1 1 1 0	1 1 1 0
couleur 14	couleur 14

POKE 49153 , 255

Inutile ici d'entrer dans les détails, les huit bits de l'octet 49153 valent 1. Le deuxième segment de l'image se met donc à clignoter avec la couleur 15 (bleu ciel sur rose).

Lignes 1, 2 et 3 : Les registres HL et DE sont chargés avec les nombres 49152 — premier segment vidéo — et 49230 — avant-dernier segment de la première ligne de l'écran. Notre nouvelle instruction provoque le transfert du contenu de l'octet numéro 49152 dans l'octet numéro 49230. Puisque nous avons écrit par POKE la valeur 63 dans l'octet 49152, cette même valeur se retrouve dans l'octet 49230. Le segment correspondant s'allume donc sur l'écran. Il clignote lui aussi en bleu sur jaune.

Ligne 4 : deuxième utilisation de LDI. L'octet pointé par HL est cette fois encore chargé dans l'octet ayant DE pour adresse. Or il faut savoir que la première instruction LDI a d'elle-même incrémenté ces deux registres. Ce qui nous conduit à la conclusion suivante : le nombre 255 (contenu de l'octet 49153) se trouve copié dans l'octet 49231. Voilà pourquoi le dernier segment de la première ligne du téléviseur s'est mis à clignoter en bleu clair sur fond rose (couleur d'encre 15). Notons que les registres HL et DE pointent maintenant sur les octets 49154 et 49232 mais que cela n'a aucune espèce d'importance pour nous, puisque notre programme s'arrête là.

Notes :

- L'instruction LDD procède de la même façon que LDI en chargeant à l'adresse pointée par DE le contenu de l'octet pointé par HL, mais effectue ensuite une décrémentation de ces deux registres.
- Lors de l'exécution des instructions LDI et LDD, le registre BC est toujours décrémenté.

CPI

Une comparaison est effectuée entre l'octet pointé par HL et l'accumulateur. Une instruction de branchement doit normalement suivre cette instruction.

Exemple : MODE D'ADRESSAGE INHÉRENT (21 octets)

Programme BASIC

```
10
20
30 FOR I = 43850 TO 43899 : POKE I,INT (RND*2) : NEXT
40 POKE 43840 , 0 : CALL 43801
50 PRINT " LE CHIFFRE 0 A ETE TIRE AU SORT " ;
70 PRINT PEEK ( 43840 ) ; " FOIS SUR 50 "
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	DD 21 40 AB		LD	IX,43840
2	21 4A AB		LD	HL,43850
3	06 32		LD	B,50
4	3E 00	DEBUT:	LD	A,0
5	ED A1		CPI	
6	20 03		JR	NZ,SUITE(+3)
7	DD 34 00		INC	(IX+0)
8	10 F5	SUITE:	DJNZ	DEBUT(−11)
9	C9		RET	

Cinquante tirages au sort ne comportant comme résultat que les valeurs 0 et 1 sont réalisés par la ligne BASIC 30. C'est le programme assembleur qui va décompter le nombre d'apparitions du chiffre 0. Pour cela, nous avons placé dans les octets 43850 à 43899 les 50 chiffres obtenus par la fonction RND.

Lignes 1 et 2 : le registre IX pointe sur l'octet 43840. Cet octet sera incrémenté à chaque fois que le chiffre 0 sera apparu ; c'est donc là que le BASIC viendra chercher la réponse finale. Nous avons pris soin naturellement d'initialiser (ligne BASIC 40) l'octet 43840 à zéro. Le registre HL, de son côté, contient l'adresse du premier octet dont on va analyser le contenu.

Lignes 4, 5 et 6 : on compare, grâce à CPI, les contenus de l'accumulateur et de l'octet 43850. Il n'y a que deux possibilités. Soit le premier nombre tiré au sort est le chiffre 1, soit c'est le chiffre 0. Dans le premier cas, la comparaison porte sur deux valeurs différentes et JR NZ branche directement l'ordinateur à la ligne SUITE. Dans le second cas, l'instruction de branchement n'a aucun effet, le programme se poursuit en séquence et le contenu de l'octet pointé par IX, l'octet 43840 donc, est incrémenté.

Ligne 8 : HL a été automatiquement incrémenté par l'instruction CPI et contient donc maintenant la valeur 43851. Au deuxième passage dans la boucle DEBUT, une nouvelle comparaison sera établie entre le contenu de l'accumulateur et de l'octet 43851. Et cela conduira à l'ajout d'une unité à l'octet 43840 si (et seulement si) le second nombre aléatoire est un 0.

Le programme se termine quand les 50 chiffres tirés au sort auront été comparés à 0.

Notes :

- L'instruction CPD est équivalente de CPI ; elle compare l'octet pointé par HL à l'accumulateur. Mais le registe HL se retrouve ensuite avec une unité de moins.
- Lors de l'exécution de ces deux instructions de comparaison, le registre BC est toujours décrémenté.

LDIR

Cette instruction programme le transfert d'une zone mémoire vers une autre zone mémoire. HL et DE pointent respectivement sur les premiers octets de chacune de ces zones. BC est chargé avec le nombre d'octets à transférer.

Exemple : MODE D'ADRESSAGE INHÉRENT (24 octets)

Programme BASIC

```
10
20
30 MODE 0 : CALL 43801 : FOR I = 1 TO 7
40 J = 49152 + 2048 * I : POKE 43851 , INT ( J / 256 )
50 POKE 43850 , J - INT ( J / 256 ) * 256
60 CALL 43812 : NEXT : LOCATE 1 , 10
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	21 00 C0		LD	HL,49152
2	06 50		LD	B,80
3	36 D8	DEBUT:	LD	(HL),216
4	23		INC	HL
5	10 FB		DJNZ	DEBUT (–5)
6	C9		RET	
7	21 00 C0		LD	HL,49152
8	ED 5B 4A AB		LD	DE,(43850)
9	01 50 00		LD	BC,80
10	ED B0		LDIR	
11	C9		RET	

Lignes 1 à 6 : la valeur 216 est écrite dans chacun des 80 premiers octets de la mémoire écran. Puisque nous sommes en mode 0, les 80 segments qui constituent la première ligne de l'image vont nous apparaître avec les couleurs rouge et noir.

En effet 216 = 11011000 binaire
ce qui donne pour les bits 1537 la valeur 0011 (couleur rouge)
et pour les bits 0426 la valeur 0101 (couleur noire)

Lignes 7 à 11 : ce sous-programme est appelé 7 fois de suite par la commande BASIC CALL 43812.

- Premier appel :

I = 1
J = 49152 + 2048 * 1 = 51200
Octet 43851 = poids fort de 51200
Octet 43850 = poids faible de 51200
Registre HL = 49152
Registre DE = 51200
Registre BC = 80

Donc, au moment où l'instruction LDIR va être exécutée, HL pointe sur le premier segment vidéo et DE sur celui qui est placé juste en dessous. Une fois le transfert réalisé, les 80 octets pointés successivement par HL (octets 49152 – 49231) ont recopié leurs contenus dans les 80 octets pointés successivement par DE (octets 51200 – 51279). La conséquence en est que la deuxième ligne de l'écran se colore elle aussi en rouge et en noir.

- Deuxième appel :

I = 2
J = 49152 + 2048 * 2 = 53248
Octet 43851 = poids fort de 53248
Octet 43850 = poids faible de 53248
Registre HL = 49152
Registre DE = 53248
Registre BC = 80

C'est la programmation d'un nouveau transfert : il concerne cette fois les zones mémoire 49152 – 49231 et 53248 – 53327. Les 80 octets de la troisième ligne de l'écran prennent alors la valeur 216, cela a pour effet de colorer les segments correspondants en rouge et noir.

- Appels suivants :

A chaque fois, une ligne se dessine sur le téléviseur. Au total nous pourrons en voir 8, toutes colorées de la même façon.

Note : LDDR n'est différente de LDIR que par le fait que les registres HL et DE sont automatiquement décrémentés.

SRA A

Tous les bits de l'accumulateur sont décalés vers la droite et le bit 0 va dans l'indicateur de retenue. Mais le bit 7 reste inchangé.

Exemple : MODE D'ADRESSAGE REGISTRE (9 octets)

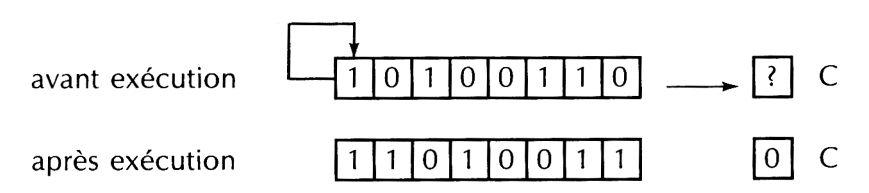
Programme BASIC

```
10
20
30 INPUT " DONNEZ UN NOMBRE NEGATIF " ; N
40 POKE 43850 , 256 + N : CALL 43801
50 PRINT " VOICI SON QUOTIENT PAR DEUX " ;
60 PRINT " – " ; 256 – PEEK ( 43860 ) : GOTO 30
```

Programme assembleur

Lignes	Codes machine	Assembleur
1	3A 4A AB	LD A,(43850)
2	CB 2F	SRA A
3	32 54 AB	LD (43860),A
4	C9	RET

Il faut rappeler que les nombres 8 bits dont l'écriture binaire commence par le chiffre 1 sont considérés par l'ordinateur comme négatifs. Par exemple –90 s'obtient en calculant le complément à deux de 90, ce qui donne 10100110. Examinons quel sera l'effet de SRA sur ce nombre si l'on suppose qu'il est écrit dans l'accumulateur :



Tous les chiffres ont été décalés vers la droite et le dernier d'entre eux est passé dans l'indicateur. Quant au bit 7, il valait 1 et, dans la place qu'il a laissée libre, le même chiffre 1 a été écrit. En se livrant au jeu des conversions, on obtient pour A la valeur décimale 211. Or, si l'on cherche le complément à deux de 45, on obtient justement 211. Ainsi, à la suite de l'exécution de SRA, l'accumulateur contient la traduction binaire de la valeur – 45. Voici donc compris le rôle de notre nouvelle instruction : elle permet de diviser par 2 un nombre négatif tout en conservant son signe. Il nous faut voir, au niveau du BASIC, par quelle gymnastique nous pouvons faire parvenir au processeur le nombre à diviser et récupérer ensuite son quotient.

N est un nombre négatif qu'il va falloir transmettre sur le mode complément à deux. Cela se fait avec le POKE de la ligne 40 : en effet, en retranchant un nombre de 256, on obtient la valeur décimale de son complément à deux. 255 correspond par exemple à – 1, 254 à – 2, etc.

On reprendra la même méthode pour traduire (ligne 60) le nombre négatif que la machine aura calculé en une forme qui nous est habituelle.

Une dernière chose : ne manquez pas de proposer à l'ordinateur des nombres impairs ou des nombres dont la valeur absolue est supérieure à 127. Et essayez de retrouver à chaque fois où se trouve la logique d'une réponse apparemment incorrecte.

DIFFÉRENTES FORMES DE L'INSTRUCTION SRA (MODE REGISTRE)

SRA	R	R est l'un des registres 8 bits.

SRA (HL)

Le contenu de l'octet mémoire pointé est soumis à une rotation sur sa droite. Le bit 7 garde sa valeur d'origine et le bit 0 passe dans l'indicateur de retenue.

Exemple : MODE D'ADRESSAGE INDIRECT (33 octets)

Programme BASIC

```
10
20
30 MODE 2 : CALL 43801
```

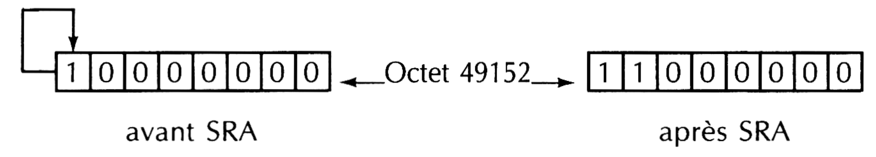
Programme assembleur

Lignes	Codes machine	Assembleur		
1	21 00 C0		LD	HL,49152
2	11 D0 07		LD	DE,2000
3	36 80	DEBUT1:	LD	(HL),128
4	23		INC	HL
5	1B		DEC	DE
6	7A		LD	A,D
7	B3		OR	E
8	20 F8		JR	NZ,DEBUT1(−8)
9	06 07		LD	B,7
10	21 00 C0	DEBUT2:	LD	HL,49152
11	11 D0 07		LD	DE,2000
12	CB 2E	SUITE:	SRA	(HL)
13	23		INC	HL
14	1B		DEC	DE
15	7A		LD	A,D
16	B3		OR	E
17	20 F8		JR	NZ,SUITE(−8)
18	10 F0		DJNZ	DEBUT2(−16)
19	C9		RET	

Lignes 1 à 8 : on écrit la valeur 128 (10000000 binaire) dans chacun des 2 000 premiers octets de la mémoire écran. Le point de gauche des segments correspondants s'allume alors en jaune. Ces segments sont situés sur 25 lignes horizontales réparties sur toute la surface de l'image.

Lignes 9 à 11 : on réinitialise les registres HL et DE avec les valeurs qu'ils contenaient au départ et on charge le registre B avec le nombre 7. Ce registre va décompter le nombre de passages dans la boucle DEBUT2.

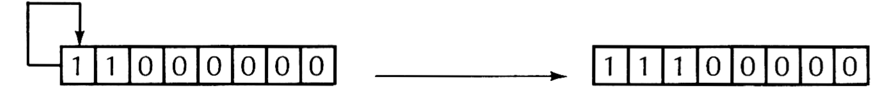
Ligne 12 : l'instruction SRA, agissant sur l'octet 49152, décale tous ses bits vers la droite et recopie dans le bit 7 le chiffre 1.



Le premier segment de l'écran nous apparaît à ce moment-là avec ses deux points de gauche allumés.

Lignes 13 à 17 : il en sera de même pour tous les autres segments pointés par HL dans la boucle SUITE.

Ligne 18 : le processeur est relancé à la ligne 10 et modifie à nouveau la configuration des 2000 premiers octets vidéo.



Les segments reliés à ces octets ont alors leurs trois points de gauche éclairés. Le programme ne se terminera que lorsque les actions répétées de SRA auront allumé tous les points de chacun des 2 000 segments. Vingt-cinq lignes complètes seront alors tracées sur l'écran.

DIFFÉRENTES FORMES DE L'INSTRUCTION SRA (MODE INDIRECT)

SRA (HL)	SRA (IX+n)	SRA (IY+n)

CPL

Le contenu de l'accumulateur est remplacé par son complément logique. Chaque chiffre 1 est transformé en un chiffre 0 et réciproquement.

Exemple : MODE D'ADRESSAGE INHÉRENT (23 octets)

Programme BASIC

```
10
20
30 MODE 2 : PRINT " INVERSION VIDEO DE LA PREMIERE LIGNE "
40 FOR I = 1 TO 50 : FOR J = 1 TO 100 : NEXT
50 CALL 43801 : NEXT
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	11 00 08		LD	DE,2048
2	21 00 C0		LD	HL,49152
3	06 24		LD	B,36
4	E5	SUITE2:	PUSH	HL
5	0E 08		LD	C,8
6	7E	SUITE1:	LD	A,(HL)
7	2F		CPL	
8	77		LD	(HL),A
9	19		ADD	HL,DE
10	0D		DEC	C
11	20 F9		JR	NZ,SUITE1(−7)
12	E1		POP	HL
13	23		INC	HL
14	10 F2		DJNZ	SUITE2(−14)
15	C9		RET	

Lignes 1 à 3 : nous utilisons une nouvelle fois la mémoire écran en faisant pointer HL sur le premier octet vidéo et en chargeant DE avec un nombre qui est l'écart entre les adresses de deux segments superposés. B contient le nombre de caractères que l'on se propose de faire clignoter.

Lignes 4 et 5 : après avoir empilé HL, on inscrit dans le registre C le nombre de passages que le programme va effectuer dans la boucle SUITE1.

Lignes 6 à 8 : on transfère dans l'accumulateur le contenu de l'octet 49152. Admettons, à titre d'exemple, que ce soit la valeur 24 (00011000 binaire) qui soit placée dans cet octet. Seuls les deux points centraux du premier segment de l'image seront alors visibles.

Faisons agir CPL :



Les bits 1 sont forcés à 0 et les bits 0 à 1. Ce qui a pour effet d'allumer les trois points de gauche et les trois points de doite du segment. On lui a donc fait subir une inversion vidéo.

Lignes 9 à 11 : on réitère l'opération sur les 7 segments placés immédiatement en dessous ; chacun d'eux va donc se voir remplacé par son complément logique. En sortant de la boucle SUITE1, la première lettre du mot INVERSION va donc être écrite sur le moniteur, non plus en jaune sur fond bleu, mais en bleu sur fond jaune.

Lignes 12 à 14 : la valeur 49152 est ressortie de la pile, réécrite dans le registre HL et aussitôt incrémentée. Et le programme se retrouve à la ligne 4. La deuxième lettre du mot INVERSION va alors elle aussi nous apparaître avec des couleurs inversées, en bleu sur jaune donc. Quand le programme assembleur arrivera à son terme, les 36 lettres de la phrase que nous avions écrite sur la première ligne de l'image auront pris des couleurs opposées.

Lorsque le BASIC reprend le contrôle du programme, il laisse s'écouler un peu de temps et "repasse la main" à l'assembleur. Nos 36 lettres retrouvent alors leur coloration d'origine. Ce que nous venons d'analyser va se reproduire sous nos yeux encore 24 fois, le temps que la boucle FOR NEXT soit exécutée complètement.

NEG

La valeur du registre A est remplacée par son complément à deux.

Exemple : MODE D'ADRESSAGE INHÉRENT (9 octets)

Programme BASIC

```
10 MEMORY 43800 : A$ = "3A4AABED44324BABC9"
20 AD = 43801 : FOR I = 0 TO 8
30 POKE AD + I , VAL("&H"+MID$(A$,2*I+1,2)) : NEXT
40 INPUT "DONNEZ UN NOMBRE " ; N
50 POKE 43850 , N : CALL 43801
60 PRINT " EN COMPLEMENT A DEUX " ; -N ;
70 PRINT " S'ECRIT " ; PEEK(43851) : GOTO 40
```

Programme assembleur

Lignes	Codes machine	Assembleur
1	3A 4A AB	LD A,(43850)
2	ED 44	NEG
3	32 4B AB	LD (43851),A
4	C9	RET

Nous voici, avec ce programme, débarrassés de tous les problèmes d'écriture des nombres négatifs sur le mode complément à deux. L'instruction NEG effectue pour nous les deux opérations nécessaires :

- complémentation logique,
- addition de 1 au résultat obtenu.

Ligne 1 : l'accumulateur est chargé avec le nombre N que le BASIC avait écrit dans l'octet 43850.

Ligne 2 : on recherche le complément à deux de N. Cette ligne aurait pu être remplacée par les deux instructions assembleur suivantes :

```
CPL
ADD A,1
```

Il ne reste plus qu'à écrire la réponse dans l'octet voulu.

On a rencontré peu de programmes machine aussi faciles à comprendre, aussi perdons un peu de temps à analyser la façon dont les codes ont été chargés par le BASIC.

Ligne BASIC 10 : la variable chaîne A$ est formée par la série des codes machine 3A, 4A, AB..., concaténés les uns aux autres.

Ligne BASIC 20 : nous retrouvons notre valeur habituelle 43801, c'est l'adresse à laquelle sera placé le premier code 3A. Une boucle FOR NEXT portant sur la variable I est alors exécutée : lorsque I vaut 0, MID$(A$,2*I+1,2) devient MID$(A$,1,2), c'est-à-dire le nombre hexadécimal 3A que POKE placera dans l'octet 43801. Puis I vaudra 1 et, cette fois, POKE inscrira le code 4A dans l'octet 43802. Cela se poursuivra jusqu'à ce que C9 soit écrit dans l'octet 43809.

Cette méthode est un peu moins lisible que celle que nous avons utilisée tout au long de ce livre mais, puisque beaucoup de programmeurs la préfèrent, autant l'avoir vue au moins une fois.

JP

Abréviation de JumP, *cette instruction permet de réaliser un branchement long à n'importe quel octet de la mémoire.*

Exemple : MODE D'ADRESSAGE ABSOLU (49 octets)

Programme BASIC

```
10
20
30 MODE 2 : PLOT 150 , 100 : CALL 43801
```

Programme assembleur

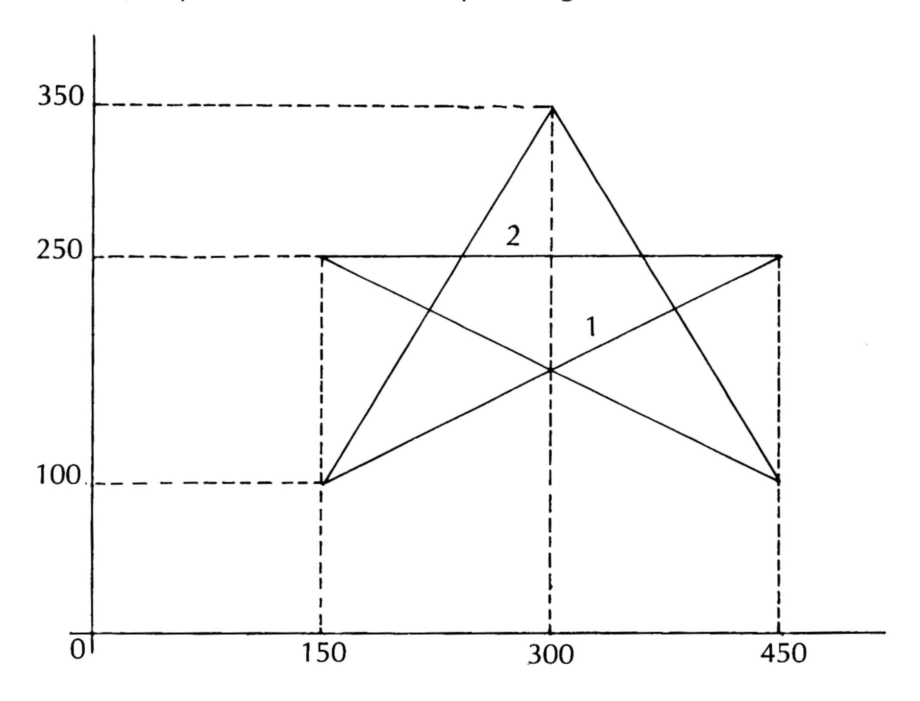
Lignes	Codes machine	Assembleur
1	C3 1C AB	JP 43804
2	11 C2 01	LD DE,450
3	21 FA 00	LD HL,250
4	CD F6 BB	CALL 48118
5	11 96 00	LD DE,150
6	21 FA 00	LD HL,250
7	CD F6 BB	CALL 48118
8	11 C2 01	LD DE,450
9	21 64 00	LD HL,100
10	CD F6 BB	CALL 48118
11	11 2C 01	LD DE,300
12	21 5E 01	LD HL,350
13	CD F6 BB	CALL 48118
14	11 96 00	LD DE,150
15	21 64 00	LD HL,100
16	CD F6 BB	CALL 48118
17	C9	RET

Ligne 1 : voici une présentation tout à fait artificielle ; elle ne sert qu'à mettre en avant la nouvelle instruction JP. Celle-ci réalise un branchement inconditionnel vers n'importe quel octet de la mémoire ; en l'occurrence, pour nous, c'est du premier octet de la ligne suivante qu'il s'agit.

Lignes 2 à 4 : le sous-programme 48118 procède, quand on l'appelle, au tracé d'une droite sur l'écran. Les points reliés sont, d'une part, le dernier point qui a été éclairé et, d'autre part, celui dont les coordonnées sont contenues par les registres DE et HL. Voyons ce que cela donne pour nous : PLOT 150 , 100 a allumé le point de coordonnées (150 , 100). DE (abscisse) et HL (ordonnée) pointent sur le point de coordonnées (450 , 250). Le segment n° 1 (voir figure) apparaît donc sur le téléviseur.

Lignes 5 à 7 : le sous-programme 48118, appelé une deuxième fois, dessine le segment n° 2. Celui-ci relie les points (450 , 250) et (150 , 250).

Lignes 8 à 17 : une fois le programme achevé, cinq lignes seront visibles, disposées comme l'indique la figure suivante :



DIFFÉRENTES FORMES DE L'INSTRUCTION DE SAUT JP

JP Z,ADRESSE	JP NZ,ADRESSE
JP C,ADRESSE	JP NC,ADRESSE
JP est utilisé dans ce cas d'une manière analogue à JR.	
JP (HL) JP (IX) JP (IY)	
Le branchement s'effectue à une adresse contenue par HL , IX ou IY.	

RET Z

Le retour au programme appelant ne se produit que lorsque l'instruction précédente est :

- *une comparaison entre deux valeurs égales,*
- *une opération donnant un résultat nul.*

Exemple : MODE D'ADRESSAGE INHÉRENT (11 octets)

Programme BASIC

```
10
20
30 MODE 0 : CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	CD 06 BB	DEBUT:	CALL	47878
2	FE 51		CP	81
3	C8		RET	Z
4	CD 5D BB		CALL	47965
5	18 F5		JR	DEBUT (−11)

Voici un programme qui va nous autoriser à imprimer sur le moniteur toutes les lettres que nous taperons au clavier.

Ligne 1 : le microprocesseur est envoyé dans la routine 47878. Il y restera tant qu'aucune touche n'aura été enfoncée. Ce sous-programme boucle donc sur lui-même en se contentant de scruter le clavier. On ne peut en sortir qu'en appuyant sur une touche.

Ligne 2 : tapons par exemple la lettre A ; rien pour autant n'apparaît sur l'écran mais cela provoque la sortie de la routine 47878. Ce qui ne nous serait d'aucune utilité si l'ordinateur n'avait eu la bonne idée de charger, de lui-même, le code de la lettre A (ASCII 65) dans l'accumulateur avant de retrouver le cours normal de notre programme. L'instruction CP compare donc les valeurs 65 et 81.

Ligne 3 : ces valeurs étant différentes, la commande RET Z est ignorée et c'est la ligne suivante qui est exécutée.

Ligne 4 : le sous-programme 47965 a déjà été mis à contribution plusieurs fois dans ce livre. Il réalise l'affichage du caractère contenu par ... l'accumulateur. C'est exactement ce qui nous convient, non ? Voilà que s'explique pourquoi la première lettre de l'alphabet se trouve maintenant dessinée en haut et à gauche de l'écran.

Ligne 5 : l'ordinateur reçoit l'ordre de se rebrancher, de façon inconditionnelle, à la première ligne. Il se replonge de ce fait dans le sous-programme de scrutation et n'en ressort que lorsqu'une touche est enfoncée. Le caractère correspondant est alors affiché. Cela durera aussi longtemps que nous n'aurons pas tapé la lettre Q (ASCII 81). Dès que cela se produira, la ligne 2 procédera à une comparaison entre deux valeurs égales et l'instruction RET Z (RETour si Zéro) nous ramènera au BASIC.

Nous aurions naturellement pu remplacer la ligne 3 par JR Z,FIN et ajouter la ligne 6 suivante : FIN: RET. Notre programme aurait fait le même travail.

DIFFÉRENTES FORMES DE L'INSTRUCTION DE RETOUR CONDITIONNEL

RET Z	RET NZ
RET C	RET NC

SET b,A

Cette instruction force à 1 le bit b de l'accumulateur.

Exemple : MODE D'ADRESSAGE REGISTRE (27 octets)

Programme BASIC

```
10
20
30 MODE 0 : CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur
1	21 0A 05	LD HL,1290
2	CD 75 BB	CALL 47989
3	3E 41	LD A,65
4	F5	PUSH AF
5	CD 5D BB	CALL 47965
6	21 14 0A	LD HL,2580
7	CD 75 BB	CALL 47989
8	F1	POP AF
9	CB CF	SET 1,A
10	CB D7	SET 2,A
11	CD 5D BB	CALL 47965
12	C9	RET

Lignes 1 et 2 : examinons la décomposition en poids fort et poids faible du registre HL :

1290 = 5 * 256 + 10

soit 5 pour le registre H et 10 pour le registre L.

Le sous-programme 47989 a pour fonction de positionner le curseur à l'intersection de la colonne et de la ligne dont les numéros sont contenus dans les registres H et L.

Lignes 3, 4 et 5 : l'accumulateur est chargé avec le code ASCII de la lettre A, ce code est sauvegardé, et la routine 47965 est appelée. Puisque son rôle est d'afficher un caractère, nous voyons apparaître sur l'écran la lettre A. Elle s'écrit à l'endroit où se trouve le curseur, cinquième colonne et dixième ligne donc.

Lignes 6 et 7 : on indique au processeur à quel autre endroit de l'image on se propose d'afficher une deuxième lettre :

2580 = 10 * 256 + 20

cela sera fait sur les dixième colonne et vingtième ligne.

Lignes 8, 9 et 10 : on retrouve la valeur d'origine de l'accumulateur, c'est-à-dire 65, et on force à 1 les bits 1 et 2 de ce nombre. A passe donc par les différentes valeurs suivantes :

ligne 8 : A = 01000001 (65 décimal)
ligne 9 : A = 01000011 (67 décimal)
ligne 10 : A = 01000111 (71 décimal)

Ligne 11 : il ne reste alors au programme d'affichage qu'à faire apparaître le caractère G (ASCII 71) à l'intersection de la colonne 10 et de la ligne 20.

DIFFÉRENTES FORMES DE L'INSTRUCTION SET

SET	b,R	b est le numéro du bit et R est l'un des registres 8 bits			
SET	b,(HL)	SET	b,(IX+n)	SET	b,(IY+n)
Le bit forcé à 1 est celui de l'octet mémoire pointé.					

RES b,(HL)

Le bit b de l'octet pointé par HL est forcé à zéro.

Exemple : MODE D'ADRESSAGE INDIRECT (22 octets)

Programme BASIC

```
10
20
30 MODE 0 : FOR I = 1 TO 20
40 FOR J = 1 TO 100 : NEXT
50 POKE 43850 , 1 : CALL 43801
60 FOR J = 1 TO 100 : NEXT
70 POKE 43850 , 0 : CALL 43801 : NEXT
```

Programme assembleur

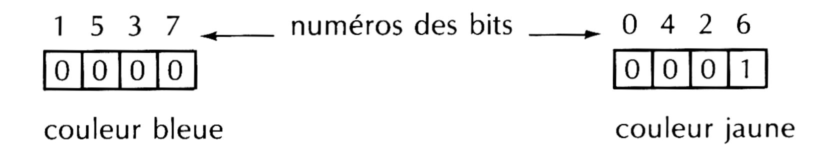
Lignes	Codes machine	Assembleur		
1	21 00 C0		LD	HL,49152
2	06 F0		LD	B,240
3	3A 4A AB	DEBUT:	LD	A,(43850)
4	FE 01		CP	1
5	28 04		JR	Z,SUITE1 (+4)
6	CB B6		RES	6,(HL)
7	18 02		JR	SUITE2 (+2)
8	CB F6	SUITE1:	SET	6,(HL)
9	23	SUITE2:	INC	HL
10	10 F0		DJNZ	DEBUT (−16)
11	C9		RET	

Le programme BASIC est constitué d'une boucle FOR NEXT I qui est exécutée vingt fois. La valeur 1 est écrite dans l'octet 43850 avant le premier appel du programme machine.

Lignes 1 et 2 : le registre B, qui va être décrémenté 240 fois, décomptera le nombre d'octets vidéo pointés successivement par HL.

Lignes 3, 4 et 5 : on compare le contenu de l'octet 43850 et le chiffre 1. Puisque les deux nombres sont identiques, le programme rejoint directement la ligne SUITE1.

Lignes 8, 9 et 10 : le bit numéro 6 de l'octet pointé par HL prend la valeur 1. L'octet 49152 passe donc de 00000000 à 01000000, et le premier segment de l'écran se colore, pour moitié en bleu et pour moitié en jaune.



Puis le registre HL est incrémenté et un deuxième passage dans la boucle DEBUT est effectué. Le contenu de l'octet 43850 n'ayant pas changé, la ligne SUITE1 est une nouvelle fois exécutée. Le deuxième segment apparaît, lui aussi coloré en bleu et jaune.
Quand le BASIC retrouve son cheminement, 240 segments bicolores peuvent se voir sur le téléviseur. C'est alors que le contenu de l'octet 43850 est abaissé à zéro et que tout recommence.

Lignes 1 à 5 : les registres B et HL sont réinitialisés. Ensuite le processeur, tenant compte du fait que l'octet 43850 n'est plus égal à 1, ne prête plus attention à l'instruction JR Z ...

Ligne 6 : ... et s'empresse de défaire ce qu'il avait fait : il annule le bit 6 de l'octet 49152. Le segment correspondant n'est alors plus visible ; il va en être de même pour ses 239 suivants quand l'instruction DJNZ sera devenue inopérante.

Vous devez maintenant avoir compris pourquoi les vingt exécutions de la boucle BASIC font apparaître puis disparaître à chaque fois 240 segments sur le moniteur.

DIFFÉRENTES FORMES DE L'INSTRUCTION RES

RES	b,R	b est le numéro du bit, R l'un des registres 8 bits.			
RES	b,(HL)	RES	b,(IX+n)	RES	b,(IY+n)
Le bit b de l'octet mémoire pointé est abaissé à 0.					

EX DE,HL

Cette instruction programme l'échange des registres DE et HL.

Exemple : MODE D'ADRESSAGE REGISTRE (19 octets)

Programme BASIC

```
10
20
30 MODE 2 : POKE 43851 , 0 : POKE 43853 ,0
40 DRAW 250,250 : FOR I = 1 TO 20
50 X = INT ( RND * 250 ) : Y = INT ( RND * 250 )
60 POKE 43850 , X : POKE 43852 , Y : CALL 43801
70 FOR J = 1 TO 1000 : NEXT : NEXT
```

Programme assembleur

Lignes	Codes machine	Assembleur	
1	ED 5B 4A AB	LD	DE,(43850)
2	2A 4C AB	LD	HL,(43852)
3	D5	PUSH	DE
4	E5	PUSH	HL
5	CD EA BB	CALL	48106
6	E1	POP	HL
7	D1	POP	DE
8	EB	EX	DE,HL
9	CD EA BB	CALL	48106
10	C9	RET	

Lignes 1 et 2 : les registres D et E sont respectivement chargés avec les valeurs des octets 43851 et 43850. Le BASIC a annulé l'octet 43851 et a écrit dans l'octet 43850 un nombre qu'il a choisi de façon aléatoire dans l'intervalle 0 – 249. Cela revient à dire que le registre DE contient le nombre X. En faisant le même raisonnement, on déduit que HL est, lui aussi, chargé avec un nombre compris entre 0 et 249.

Lignes 3 à 7 : la précaution d'empiler DE et HL est prise car l'appel de la ligne 5 ''corrompt'' ces deux registres. CALL 48106 a pour effet, nous le savons, d'allumer un point sur l'écran. Il va apparaître en jaune à l'intersection de la colonne X et de la ligne Y. Afin de mieux comprendre la suite, nous supposerons que DE est égal à 100 et HL à 200. Le point dont il est question a donc pour coordonnées (100,200).

Ligne 8 : l'instruction d'échange, très simple à utiliser, recopie dans DE et HL les valeurs 200 et 100.

Ligne 9 : le deuxième appel du sous-programme d'affichage graphique rend visible un point ayant pour coordonnées (200,100). Il est très exactement symétrique du précédent par rapport à la ''diagonale montante'' du téléviseur. Cette diagonale a d'ailleurs été matérialisée par la commande DRAW de la ligne 40.

Le tracé que nous venons d'analyser va être renouvelé vingt fois, à l'intérieur de la boucle BASIC FOR NEXT I. Quarante points, deux à deux symétriques par rapport à la diagonale, seront alors apparus sur l'écran.

DIFFÉRENTES FORMES DE L'INSTRUCTION D'ÉCHANGE

EX DE,HL
EX (SP),HL EX (SP),IX EX (SP),IY L'échange est réalisé entre la mémoire pointée par SP et l'un des registres HL, IX ou IY.

CONCLUSION

Ce livre a constitué une introduction à la programmation en langage machine de l'ordinateur Amstrad. Nous en avons étudié les aspects les plus importants et réalisé une série d'exercices qui vous ont montré, c'est notre souhait, que l'assembleur pouvait être assimilé sans difficulté par un lecteur armé de sa seule bonne volonté. Nous sommes persuadés, pour notre part, qu'il est infiniment plus long d'acquérir la logique de la programmation BASIC que celle de l'assembleur.

Vous êtes maintenant en mesure de créer vos propres programmes et d'inclure dans vos lignes BASIC des effets spéciaux que seule l'impressionnante rapidité de l'assembleur autorise. Si l'occasion se présente, vous ne manquerez pas de chercher à quoi correspondent les codes machine que d'autres programmeurs auront obtenus, faisant ainsi le travail inverse de celui qui a été effectué jusqu'à maintenant. Cette opération, qui s'appelle le désassemblage, vous permettra de reconstruire le programme assembleur et éventuellement de le modifier pour qu'il s'adapte très précisément à votre cas.

Naturellement, rien ne vous empêche de franchir une nouvelle étape en vous orientant vers des ouvrages plus techniques que celui-ci[1]. Vous y trouverez des programmes applicables à la gestion des périphériques ainsi que des explications concernant les quelques instructions que nous avons volontairement passées sous silence, estimant que, dans un premier temps en tout cas, leur intérêt était négligeable.

Il se pourrait que vous ressentiez maintenant la nécessité de vous procurer la cassette Zen contenant le programme éditeur/assembleur de l'Amstrad. Elle réalisera pour vous, sans risque d'erreur et très rapidement, la traduction en langage machine des programmes écrits en assembleur. C'est l'auxiliaire indispensable de tous ceux qui ont découvert, avec passion, que l'on pouvait s'adresser directement à un microprocesseur.

Ce livre s'achève sur trois programmes un peu plus compliqués que les autres. Vous les aborderez sans complexe maintenant que vous est ouvert l'étroit mais ô combien royal chemin de l'assembleur.

1. ***Programmation du Z80*** par Rodnay Zaks.

ANNEXE A
DÉPLACEMENT D'UN MOBILE SUR L'ÉCRAN

Programme BASIC (65 octets)

```
 10
 20
 30 MODE 2 : SYMBOL 255,224,240,127,31,31,127,240,224
 40 SYMBOL 254,0,0,0,255,255 : SYMBOL 253,0,128,99,31,12,7,8
 50 SYMBOL 252,0,0,255,36,219,255 : SYMBOL 251,0,1,198,248,48,224,16
 60 LOCATE 1,11 : PRINT CHR$(255) ; CHR$(254)
 70 FOR I = 1 TO INT(RND*3000) : NEXT
 80 FOR Y = 2 TO 20 : LOCATE 78,Y-1 : PRINT "   "
 90 LOCATE 78,Y : PRINT CHR$(253) ; CHR$(252) ; CHR$(251)
100 FOR I = 1 TO 10 : NEXT : IF INKEY$ < > "" THEN 120
110 NEXT : LOCATE 78,20 : PRINT "   " : GOTO 60
120 CALL 43801 : LOCATE 40,10
130 IF Y = 11 THEN PRINT "GAGNE" ELSE PRINT "PERDU"
140 FOR I = 1 TO 1000 : NEXT : LOCATE 40,10
150 PRINT "      " : LOCATE 78,Y : PRINT "   " : GOTO 60
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	11 00 08		LD	DE,2048
2	DD 21 20 C3		LD	IX,49952
3	06 4E		LD	B,78
4	OE 08	DEBUT:	LD	C,8
5	DD E5	CASE:	PUSH	IX
6	3E 08		LD	A,8
7	DD CB 00 3E	SUITE	SRL	(IX+0)
8	DD CB 01 1E		RR	(IX+1)
9	DD CB 02 1E		RR	(IX+2)
10	DD 19		ADD	IX,DE
11	3D		DEC	A
12	20 EF		JR	NZ,SUITE
13	CD 54 AB		CALL	TEMPO
14	DD E1		POP	IX
15	0D		DEC	C
16	20 E3		JR	NZ,CASE
17	DD 23		INC	IX
18	10 DD		DJNZ	DEBUT
19	06 08		LD	B,8
20	DD 36 00 00	EFFACE:	LD	(IX+0),0
21	DD 36 01 00		LD	(IX+1),0
22	DD 19		ADD	IX,DE
23	10 F4		DJNZ	EFFACE
24	C9		RET	
25	3E 50	TEMPO:	LD	A,80
26	3D	ATTENTE:	DEC	A
27	20 FD		JR	NZ,ATTENTE
28	C9		RET	

Le but de ce programme est de déplacer un missile à travers l'écran pour atteindre une cible qui descend à droite du moniteur.

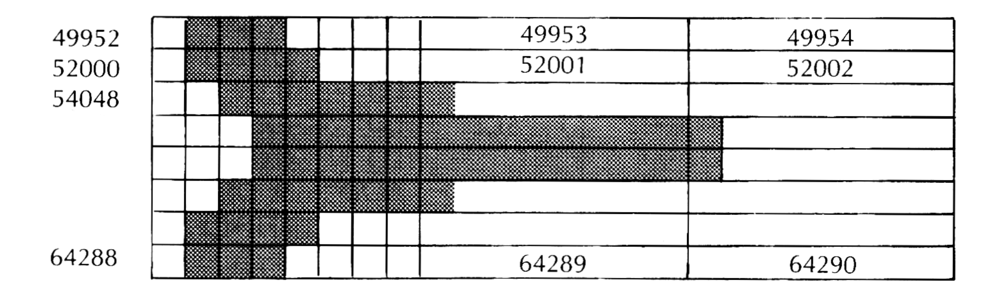
49952 | 49953 | 49954
52000 | 52001 | 52002
54048
64288 | 64289 | 64290

Lignes 7 à 16 : chacun des octets 49952, 52000 ... 64288 est décalé vers la droite. Les bits 7 sont tous remplacés par des zéros et les bits 0 passent dans l'indicateur de retenue.

En même temps, les octets 49953, 52001 ... 64289 voient leurs contenus subir une rotation vers la droite. Les bits 7 sont remplacés par l'indicateur de retenue et les bits 0 entrent dans cet indicateur.

C'est le même processus qui est réalisé pour les octets 49954, 52002 ... 64290.

Voici alors ce qu'est devenu notre missile :



Il a été entièrement translaté d'une position sur la droite. Puisque cette transformation se reproduit huit fois de suite (compteur C), nous le retrouvons déplacé sur sa droite d'une distance égale à la largeur d'un caractère en mode 2.

Lignes 17 et 18 : IX est incrémenté, prend la valeur 49953 et le programme est rebranché à la ligne 4. Le missile va, cette fois encore, être déplacé de huit positions élémentaires sur sa droite. Sa partie ailerons se trouve alors sur la troisième colonne et son fuselage sur la quatrième.

Quand le programme arrive à son terme (compteur B à zéro), le missile est dessiné sur les deux dernières cases de la ligne texte numéro 11.

Lignes 19 à 24 : il ne reste plus qu'à l'effacer avant de ''rendre la main'' au BASIC. Cela est réalisé en écrivant la valeur 0 dans les 16 octets qui contiennent le dessin du mobile.

ANNEXE B
TRI
EN MÉMOIRE CENTRALE

Programme BASIC (27 octets)

```
10
20
30 FOR I = 43850 TO 43899 : X = INT (RND*250)
40 PRINT X ; : POKE I , X : NEXT : PRINT
50 CALL 43801 : FOR I = 43850 TO 43899
60 PRINT PEEK (I) ; : NEXT
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1	21 4A AB		LD	HL,43850
2	06 31		LD	B,49
3	C5	DEBUT:	PUSH	BC
4	54		LD	D,H
5	5D		LD	E,L
6	13		INC	DE
7	1A	PASSE:	LD	A,(DE)
8	BE		CP	(HL)
9	30 06		JR	NC,RIEN
10	1A		LD	A,(DE)
11	F5		PUSH	AF
12	7E		LD	A,(HL)
13	12		LD	(DE),A
14	F1		POP	AF
15	77		LD	(HL),A
16	13	RIEN:	INC	DE
17	10 F3		DJNZ	PASSE
18	23		INC	HL
19	C1		POP	BC
20	10 EB		DJNZ	DEBUT
21	C9		RET	

Le programme BASIC place dans les octets dont les adresses s'échelonnent entre 43850 et 43899, cinquante valeurs tirées au sort. L'assembleur a la charge de mettre de l'ordre dans cette liste et de retourner les 50 octets rangés par ordre croissant.

Lignes 1 et 2 : le registre HL est chargé avec l'adresse du premier élément de la liste et B avec une valeur qui est égale au nombre de passages que le programme va effectuer dans la boucle DEBUT.

Lignes 4 à 6 : on écrit dans DE le nombre 43851...

Lignes 7 à 9 : ... et on compare les contenus des octets 43851 et 43850.

Lignes 10 à 15 : si le deuxième élément de la suite est supérieur au premier, aucune action n'est faite et le programme poursuit son déroulement à la ligne RIEN. Dans le cas contraire, on procède à une permutation entre les contenus des octets 43850 et 43851. On réalise cet échange en trois phases :

(43851) ——→ Pile
(43850) ——→ A ——→ (43851)
Pile ——→ A ——→ (43850)

Lignes 16 et 17 : nous sommes maintenant certains que le premier nombre est inférieur ou égal au deuxième.
L'incrémentation de DE fait que ce registre pointe dès lors sur le troisième élément de la série. Quand l'ordinateur sera passé pour la deuxième fois dans la boucle PASSE, nous serons en mesure d'affirmer que le premier élément est aussi inférieur au troisième. Au 49e passage, nous aurons la certitude que l'élément numéro un est inférieur à tous les autres.

Lignes 18 à 20 : on fait pointer HL sur le deuxième nombre de la liste et on relance le programme à la ligne 3. On compare successivement le deuxième élément à ses 48 suivants, en effectuant un échange quand ces derniers lui sont inférieurs. A la fin de cette étape, les deux premiers nombres sont plus petits que tous les autres et ils sont eux-mêmes rangés dans l'ordre croissant.

Il reste alors à comparer le troisième, le quatrième ... aux nombres qui les suivent dans la liste.

ANNEXE C
PROGRAMMATION DU GÉNÉRATEUR DE SON

Programme BASIC (44 octets)

```
10
20
30 CALL 43801
```

Programme assembleur

Lignes	Codes machine	Assembleur		
1			ORG	43801
2			LOAD	43801
3	06 08		LD	B,8
4	21 34 AB		LD	HL,TABLE1 ; 43828
5	11 3D AB		LD	DE,TABLE2 ; 43837
6	1A	SUITE:	LD	A,(DE)
7	32 37 AB		LD	(TABLE1+3),A ; 43831
8	C5		PUSH	BC
9	E5		PUSH	HL
10	D5		PUSH	DE
11	CD AA BC		CALL	48298
12	D1		POP	DE
13	E1		POP	HL
14	C1		POP	BC
15	30 F1		JR	NC,SUITE
16	13		INC	DE
17	10 EE		DJNZ	SUITE
18	C9		RET	
19	01	TABLE1:	DB	1 ; CANAL 1
20	00		DB	0 ; VOLUME
21	00		DB	0 ; ENVELOPPE TON
22	77		DB	119 ; PERIODE
23	00		DB	0 ; PERIODE
24	00		DB	0 ; BRUITAGE
25	0C		DB	12 ; AMPLITUDE INIT.
26	14		DB	20 ; DUREE
27	00		DB	0 ; DUREE
28	77	TABLE2:	DB	119 ; DO
29	6A		DB	106 ; RE
30	5F		DB	95 ; MI
31	59		DB	89 ; FA
32	50		DB	80 : SOL
33	47		DB	71 ; LA
34	3F		DB	63 ; SI
35	3C		DB	60 ; DO
36			END	

Lorsque ce programme est exécuté, l'ordinateur nous fait entendre une gamme de huit notes. La mise au point a été réalisée avec la cassette Zen, mais le lecteur qui ne la possède pas recopiera, comme à l'accoutumée, la série des codes machine dans des lignes de DATA et les chargera par POKE dans les 44 octets de la plage mémoire 43801 – 43844.

Lignes 3, 4 et 5 : on écrit dans HL le nombre 43828 ; c'est l'adresse de la première donnée de la table 1. Puis on charge DE avec l'adresse de la deuxième table. B, pour sa part, va décompter le nombre de notes jouées.

Lignes 6 et 7 : le contenu de l'octet 43837, le nombre 119, transite par A et est recopié dans l'octet 43831. Notons que le programme assembleur/éditeur nous a en réalité débarrassés de tous ces calculs. Nous indiquons au microprocesseur qu'il doit charger l'octet TABLE1 + 3 avec la valeur 119, sans nous préoccuper de savoir quelle est l'adresse effective de cet octet.

Ligne 11 : CALL 48298 appelle le sous-programme de génération de son. Avant cet appel, HL a été positionné pour qu'il pointe sur le premier octet de la série TABLE1. Cette suite de 9 octets indique les caractéristiques du son qui va être émis. C'est un DO (période 119) d'une durée de vingt centièmes de seconde. Son amplitude initiale est de niveau 12 et il est entendu grâce au canal 1. Nous voyons que deux octets sont nécessaires pour la période et la durée car ces données peuvent prendre des valeurs supérieures à 255.

Lignes 16 et 17 : DE est incrémenté et contient alors le nombre 43838. Dans l'octet correspondant se trouve la période de la deuxième note que nous souhaitons faire jouer par l'ordinateur. Quand le programme se rebranche à la ligne 6, le contenu de l'octet 43831 devient égal à 106 et c'est la note RE qui est alors entendue. Remarquez que HL pointe pendant toute la durée du programme sur l'octet TABLE1 et que nous ne modifions que l'octet TABLE1 + 3.

Ligne 18 : quand la commande RET sera exécutée, les huit notes de la gamme auront été jouées. Elles ont toutes les mêmes caractéristiques car nous ne sommes intervenus que sur la valeur de la période.

Notes :

- L'instruction DB des lignes 19 à 35 définit le contenu d'un octet. Elle ne fait rien d'autre que d'écrire dans l'octet courant le nombre placé à sa droite.
- L'instruction JR NC,SUITE de la ligne 15 est là pour empêcher le Z80 de ''marcher plus vite que la musique''. En effet, si on la supprime, la totalité de la gamme n'est pas entendue. Cela provient du fait, mais vous le savez, que les notes sont mises en file d'attente (la queue) avant d'être jouées et que cette file n'est pas très grande. Au retour du sous-programme 48298, l'indicateur C est positionné à 0 si la file est déjà complète. Il faut alors retourner voir si, entre-temps, une place ne se serait pas libérée.

ANNEXE D
JEU D'INSTRUCTIONS DU Z80

CODE OBJET	INSTRUCTION	
8E	ADC	A,(HL)
DD8E05	ADC	A,(IX+d)
FD8E05	ADC	A,(IY+d)
8F	ADC	A,A
88	ADC	A,B
89	ADC	A,C
8A	ADC	A,D
8B	ADC	A,E
8C	ADC	A,H
8D	ADC	A,L
CE20	ADC	A,n
ED4A	ADC	HL,BC
ED5A	ADC	HL,DE
ED6A	ADC	HL,HL
ED7A	ADC	HL,SP
86	ADD	A,(HL)
DD8605	ADD	A,(IX+d)
FD8605	ADD	A,(IY+d)
87	ADD	A,A
80	ADD	A,B
81	ADD	A,C
82	ADD	A,D
83	ADD	A,E
84	ADD	A,H
85	ADD	A,L
C620	ADD	A,n
09	ADD	HL,BC
19	ADD	HL,DE
29	ADD	HL,HL
39	ADD	HL,SP
DD09	ADD	IX,BC
DD19	ADD	IX,DE
DD29	ADD	IX,IX
DD39	ADD	IX,SP
FD09	ADD	IY,BC
FD19	ADD	IY,DE
FD29	ADD	IY,IY
FD39	ADD	IY,SP
A6	AND	(HL)
DDA605	AND	(IX+d)
FDA605	AND	(IY+d)
A7	AND	A
A0	AND	B
A1	AND	C
A2	AND	D
A3	AND	E
A4	AND	H
A5	AND	L
E620	AND	n
CB46	BIT	0,(HL)
DDCB0546	BIT	0,(IX+d)
FDCB0546	BIT	0,(IY+d)
CB47	BIT	0,A
CB40	BIT	0,B
CB41	BIT	0,C
CB42	BIT	0,D
CB43	BIT	0,E
CB44	BIT	0,H
CB45	BIT	0,L
CB4E	BIT	1,(HL)
DDCB054E	BIT	1,(IX+d)
FDCB054E	BIT	1,(IY+d)
CB4F	BIT	1,A
CB48	BIT	1,B
CB49	BIT	1,C
CB4A	BIT	1,D
CB4B	BIT	1,E
CB4C	BIT	1,H
CB4D	BIT	1,L
CB56	BIT	2,(HL)
DDCB0556	BIT	2,(IX+d)
FDCB0556	BIT	2,(IY+d)
CB57	BIT	2,A
CB50	BIT	2,B
CB51	BIT	2,C
CB52	BIT	2,D
CB53	BIT	2,E
CB54	BIT	2,H
CB55	BIT	2,L
CB5E	BIT	3,(HL)
DDCB055E	BIT	3,(IX+d)
FDCB055E	BIT	3,(IY+d)
CB5F	BIT	3,A
CB58	BIT	3,B
CB59	BIT	3,C
CB5A	BIT	3,D
CB5B	BIT	3,E
CB5C	BIT	3,H
CB5D	BIT	3,L
CB66	BIT	4,(HL)
DDCB0566	BIT	4,(IX+d)
FDCB0566	BIT	4,(IY+d)
CB67	BIT	4,A
CB60	BIT	4,B
CB61	BIT	4,C
CB62	BIT	4,D

CODE OBJET	INSTRUCTION	
CB63	BIT	4,E
CB64	BIT	4,H
CB65	BIT	4,L
CB6E	BIT	5,(HL)
DDCB056E	BIT	5,(IX+d)
FDCB056E	BIT	5,(IY+d)
CB6F	BIT	5,A
CB68	BIT	5,B
CB69	BIT	5,C
C86A	BIT	5,D
CB6B	BIT	5,E
CB6C	BIT	5,H
CB6D	BIT	5,L
CB76	BIT	6,(HL)
DDCB0576	BIT	6,(IX+d)
FDCB0576	BIT	6,(IY+d)
CB77	BIT	6,A
CB70	BIT	6,B
CB71	BIT	6,C
CB72	BIT	6,D
CB73	BIT	6,E
CB74	BIT	6,H
CB75	BIT	6,L
CB7E	BIT	7,(HL)
DDCB057E	BIT	7,(IX+d)
FDCB057E	BIT	7,(IY+d)
CB7F	BIT	7,A
CB78	BIT	7,B
CB79	BIT	7,C
CB7A	BIT	7,D
CB7B	BIT	7,E
CB7C	BIT	7,H
CB7D	BIT	7,L
DC8405	CALL	C,nn
FC8405	CALL	M,nn
D48405	CALL	NC,nn
C48405	CALL	NZ,nn
F48405	CALL	P,nn
EC8405	CALL	PE,nn
E48405	CALL	PO,nn
CC8405	CALL	Z,nn
CD8405	CALL	nn
3F	CCF	
BE	CP	(HL)
DDBE05	CP	(IX+d)
FDBE05	CP	(IY+d)
BF	CP	A
B8	CP	B
B9	CP	C
BA	CP	D
BB	CP	E
BC	CP	H
BD	CP	L
FE20	CP	n
EDA9	CPD	
EDB9	CPDR	

CODE OBJET	INSTRUCTION	
EDB1	CPIR	
EDA1	CPI	
2F	CPL	
27	DAA	
35	DEC	(HL)
DD3505	DEC	(IX+d)
FD3505	DEC	(IY+d)
3D	DEC	A
05	DEC	B
0B	DEC	BC
0D	DEC	C
15	DEC	D
1B	DEC	DE
1D	DEC	E
25	DEC	H
2B	DEC	HL
DD2B	DEC	IX
FD2B	DEC	IY
2D	DEC	L
3B	DEC	SP
F3	DI	
102E	DJNZ	e
FB	EI	
E3	EX	(SP),HL
DDE3	EX	(SP),IX
FDE3	EX	(SP),IY
08	EX	AF,AF'
EB	EX	DE,HL
D9	EXX	
76	HALT	
ED46	IM	0
ED56	IM	1
ED5E	IM	2
ED78	IN	A,(C)
ED40	IN	B,(C)
ED48	IN	C,(C)
ED50	IN	D,(C)
ED58	IN	E,(C)
ED60	IN	H,(C)
ED68	IN	L,(C)
34	INC	(HL)
DD3405	INC	(IX+d)
FD3405	INC	(IY+d)
3C	INC	A
04	INC	B
03	INC	BC
0C	INC	C
14	INC	D
13	INC	DE
1C	INC	E
24	INC	H
23	INC	HL
DD23	INC	IX
FD23	INC	IY
2C	INC	L
33	INC	SP
DB20	IN	A,(n)

CODE OBJET	INSTRUCTION	
EDAA	IND	
EDBA	INDR	
EDA2	INI	
EDB2	INIR	
C38405	JP	nn
E9	JP	(HL)
DDE9	JP	(IX)
FDE9	JP	(IY)
DA8405	JP	C,nn
FA8405	JP	M,nn
D28405	JP	NC,nn
C28405	JP	NZ,nn
F28405	JP	P,nn
EA8405	JP	PE,nn
E28405	JP	PO,nn
CA8405	JP	Z,nn
382E	JR	C,e
302E	JR	NC,e
202E	JR	NZ,e
282E	JR	Z,e
182E	JR	e
02	LD	(BC),A
12	LD	(DE),A
77	LD	(HL),A
70	LD	(HL),B
71	LD	(HL),C
72	LD	(HL),D
73	LD	(HL),E
74	LD	(HL),H
75	LD	(HL),L
3620	LD	(HL),n
DD7705	LD	(IX+d),A
DD7005	LD	(IX+d),B
DD7105	LD	(IX+d),C
DD7205	LD	(IX+d),D
DD7305	LD	(IX+d),E
DD7405	LD	(IX+d),H
DD7505	LD	(IX+d),L
DD360520	LD	(IX+d),n
FD7705	LD	(IY+d),A
FD7005	LD	(IY+d),B
FD7105	LD	(IY+d),C
FD7205	LD	(IY+d),D
FD7305	LD	(IY+d),E
FD7405	LD	(IY+d),H
FD7505	LD	(IY+d),L
FD360520	LD	(IY+d),n
328405	LD	(nn),A
ED438405	LD	(nn),BC
ED538405	LD	(nn),DE
228405	LD	(nn),HL
DD228405	LD	(nn),IX
FD228405	LD	(mn),IY
ED738405	LD	(nn),SP
0A	LD	A,(BC)
1A	LD	A,(DE)
7E	LD	A,(HL)

CODE OBJET	INSTRUCTION	
DD7E05	LD	A,(IX+d)
FD7E05	LD	A,(IY+d)
3A8405	LD	A,(nn)
7F	LD	A,A
78	LD	A,B
79	LD	A,C
7A	LD	A,D
7B	LD	A,E
7C	LD	A,H
ED57	LD	A,I
7D	LD	A,L
3E20	LD	A,n
ED5F	LD	A,R
46	LD	B,(HL)
DD4605	LD	B,(IX+d)
FD4605	LD	B,(IY+d)
47	LD	B,A
40	LD	B,B
41	LD	B,C
42	LD	B,D
43	LD	B,E
44	LD	B,H
45	LD	B,L
0620	LD	B,n
ED4B8405	LD	BC,(nn)
018405	LD	BC,nn
4E	LD	C,(HL)
DD4E05	LD	C,(IX+d)
FD4E05	LD	C,(IY+d)
4F	LD	C,A
48	LD	C,B
49	LD	C,C
4A	LD	C,D
4B	LD	C,E
4C	LD	C,H
4D	LD	C,L
0E20	LD	C,n
56	LD	D,(HL)
DD5605	LD	D,(IX+d)
FD5605	LD	D,(IY+d)
57	LD	D,A
50	LD	D,B
51	LD	D,C
52	LD	D,D
53	LD	D,E
54	LD	D,H
55	LD	D,L
1620	LD	D,n
ED5B8405	LD	DE,(nn)
118405	LD	DE,nn
5E	LD	E,(HL)
DD5E05	LD	E,(IX+d)
FD5E05	LD	E,(IY+d)
5F	LD	E,A
58	LD	E,B
59	LD	E,C
5A	LD	E,D

CODE OBJET	INSTRUCTION	
5B	LD	E,E
5C	LD	E,H
5D	LD	E,L
1E20	LD	E,n
66	LD	H,(HL)
DD6605	LD	H,(IX+d)
FD6605	LD	H,(IY+d)
67	LD	H,A
60	LD	H,B
61	LD	H,C
62	LD	H,D
63	LD	H,E
64	LD	H,H
65	LD	H,L
2620	LD	H,n
2A8405	LD	HL,(nn)
218405	LD	HL,nn
ED47	LD	I,A
DD2A8405	LD	IX,(nn)
DD218405	LD	IX,nn
FD2A8405	LD	IY,(nn)
FD218405	LD	IY,nn
6E	LD	L,(HL)
DD6E05	LD	L,(IX+d)
FD6E05	LD	L,(IY+d)
6F	LD	L,A
68	LD	L,B
69	LD	L,C
6A	LD	L,D
6B	LD	L,E
6C	LD	L,H
6D	LD	L,L
2E20	LD	L,n
ED4F	LD	R,A
ED7B8405	LD	SP,(nn)
F9	LD	SP,HL
DDF9	LD	SP,IX
FDF9	LD	SP,IY
318405	LD	SP,nn
EDA8	LDD	
EDB8	LDDR	
EDA0	LDI	
EDB0	LDIR	
ED44	NEG	
00	NOP	
B6	OR	(HL)
DDB605	OR	(IX+d)
FDB605	OR	(IY+d)
B7	OR	A
B0	OR	B
B1	OR	C
B2	OR	D
B3	OR	E
B4	OR	H
B5	OR	L
F620	OR	n
ED8B	OTDR	
EDB3	OTIR	
ED79	OUT	(C),A
ED41	OUT	(C),B
ED49	OUT	(C),C
ED51	OUT	(C),D
ED59	OUT	(C),E
ED61	OUT	(C),H
ED69	OUT	(C),L
D320	OUT	(n),A
EDAB	OUTD	
EDA3	OUTI	
F1	POP	AF
C1	POP	BC
D1	POP	DE
E1	POP	HL
DDE1	POP	IX
FDE1	POP	IY
F5	PUSH	AF
C5	PUSH	BC
D5	PUSH	DE
E5	PUSH	HL
DDE5	PUSH	IX
FDE5	PUSH	IY
CB86	RES	0,(HL)
DDCB0586	RES	0,(IX+d)
FDCB0586	RES	0,(IY+d)
CB87	RES	0,A
CB80	RES	0,B
CB81	RES	0,C
CB82	RES	0,D
CB83	RES	0,E
CB84	RES	0,H
CB85	RES	0,L
CB8E	RES	1,(HL)
DDCB058E	RES	1,(IX+d)
FDCB058E	RES	1,(IY+d)
CB8F	RES	1,A
CB88	RES	1,B
CB89	RES	1,C
CB8A	RES	1,D
CB8B	RES	1,E
CB8C	RES	1,H
CB8D	RES	1,L
CB96	RES	2,(HL)
DDCB0596	RES	2,(IX+d)
FDCB0596	RES	2,(IY+d)
CB97	RES	2,A
CB90	RES	2,B
CB91	RES	2,C
CB92	RES	2,D
CB93	RES	2,E
CB94	RES	2,H
CB95	RES	2,L
CB9E	RES	3,(HL)
DDCB059E	RES	3,(IX+d)
FDCB059E	RES	3,(IY+d)

CODE OBJET	INSTRUCTION	
CB9F	RES	3,A
CB98	RES	3,B
CB99	RES	3,C
CB9A	RES	3,D
CB9B	RES	3,E
CB9C	RES	3,H
CB9D	RES	3,L
CBA6	RES	4,(HL)
DDCB05A6	RES	4,(IX+d)
FDCB05A6	RES	4,(IY+d)
CBA7	RES	4,A
CBA0	RES	4,B
CBA1	RES	4,C
CBA2	RES	4,D
CBA3	RES	4,E
CBA4	RES	4,H
CBA5	RES	4,L
CBAE	RES	5,(HL)
DDCB05AE	RES	5,(IX+d)
FDCB05AE	RES	5,(IY+d)
CBAF	RES	5,A
CBA8	RES	5,B
CBA9	RES	5,C
CBAA	RES	5,D
CBAB	RES	5,E
CBAC	RES	5,H
CBAD	RES	5,L
CBB6	RES	6,(HL)
DDCB05B6	RES	6,(IX+d)
FDCB05B6	RES	6,(IY+d)
CBB7	RES	6,A
CBB0	RES	6,B
CBB1	RES	6,C
CBB2	RES	6,D
CBB3	RES	6,E
CBB4	RES	6,H
CBB5	RES	6,L
CBBE	RES	7,(HL)
DDCB05BE	RES	7,(IX+d)
FDCB05BE	RES	7,(IY+d)
CBBF	RES	7,A
CBB8	RES	7,B
CBB9	RES	7,C
CBBA	RES	7,D
CBBB	RES	7,E
CBBC	RES	7,H
CBBD	RES	7,L
C9	RET	
D8	RET	C
F8	RET	M
D0	RET	NC
C0	RET	NZ
F0	RET	P
E8	RET	PE
E0	RET	PO
C8	RET	Z

CODE OBJET	INSTRUCTION	
ED4D	RETI	
ED45	RETN	
CB16	RL	(HL)
DDCB0516	RL	(IX+d)
FDCB0516	RL	(IY+d)
CB17	RL	A
CB10	RL	B
CB11	RL	C
CB12	RL	D
CB13	RL	E
CB14	RL	H
CB15	RL	L
17	RLA	
CB06	RLC	(HL)
DDCB0506	RLC	(IX+d)
FDCB0506	RLC	(IY+d)
CB07	RLC	A
CB00	RLC	B
CB01	RLC	C
CB02	RLC	D
CB03	RLC	E
CB04	RLC	H
CB05	RLC	L
07	RLCA	
ED6F	RLD	
CB1E	RR	(HL)
DDCB051E	RR	(IX+d)
FDCB051E	RR	(IY+d)
CB1F	RR	A
CB18	RR	B
CB19	RR	C
CB1A	RR	D
CB1B	RR	E
CB1C	RR	H
CB1D	RR	L
1F	RRA	
CB0E	RRC	(HL)
DDCB050E	RRC	(IX+d)
FDCB050E	RRC	(IY+d)
CB0F	RRC	A
CB08	RRC	B
CB09	RRC	C
CB0A	RRC	D
CB0B	RRC	E
CB0C	RRC	H
CB0D	RRC	L
OF	RRCA	
ED67	RRD	
C7	RST	00H
CF	RST	08H
D7	RST	10H
DF	RST	18H
E7	RST	20H
EF	RST	28H
F7	RST	30H
FF	RST	38H
DE20	SBC	A,n

CODE OBJET	INSTRUCTION	
9E	SBC	A,(HL)
DD9E05	SBC	A,(IX+d)
FD9E05	SBC	A,(IY+d)
9F	SBC	A,A
98	SBC	A,B
99	SBC	A,C
9A	SBC	A,D
9B	SBC	A,E
9C	SBC	A,H
9D	SBC	A,L
ED42	SBC	HL,BC
ED52	SBC	HL,DE
ED62	SBC	HL,HL
ED72	SBC	HL,SP
37	SCF	
CBC6	SET	0,(HL)
DDCB05C6	SET	0,(IX+d)
FDCB05C6	SET	0,(IY+d)
CBC7	SET	0,A
CBC0	SET	0,B
CBC1	SET	0,C
CBC2	SET	0,D
CBC3	SET	0,E
CBC4	SET	0,H
CBC5	SET	0,L
CBCE	SET	1,(HL)
DDCB05CE	SET	1,(IX+d)
FDCB05CE	SET	1,(IY+d)
CBCF	SET	1,A
CBC8	SET	1,B
CBC9	SET	1,C
CBCA	SET	1,D
CBCB	SET	1,E
CBCC	SET	1,H
CBCD	SET	1,L
CBD6	SET	2,(HL)
DDCB05D6	SET	2,(IX+d)
FDCB05D6	SET	2,(IY+d)
CBD7	SET	2,A
CBD0	SET	2,B
CBD1	SET	2,C
CBD2	SET	2,D
CBD3	SET	2,E
CBD4	SET	2,H
CBD5	SET	2,L
CBD8	SET	3,B
CBDE	SET	3,(HL)
DDCB05DE	SET	3,(IX+d)
FDCB05DE	SET	3,(IY+d)
CBDF	SET	3,A
CBD9	SET	3,C
CBDA	SET	3,D
CBDB	SET	3,E
CBDC	SET	3,H
CBDD	SET	3,L
CBE6	SET	4,(HL)

CODE OBJET	INSTRUCTION	
DDCB05E6	SET	4,(IX+d)
FDCB05E6	SET	4,(IY+d)
CBE7	SET	4,A
CBE0	SET	4,B
CBE1	SET	4,C
CBE2	SET	4,D
CBE3	SET	4,E
CBE4	SET	4,H
CBE5	SET	4,L
CBEE	SET	5,(HL)
DDCB05EE	SET	5,(IX+d)
FDCB05EE	SET	5,(IY+d)
CBEF	SET	5,A
CBE8	SET	5,B
CBE9	SET	5,C
CBEA	SET	5,D
CBEB	SET	5,E
CBEC	SET	5,H
CBED	SET	5,L
CBF6	SET	6,(HL)
DDCB05F6	SET	6,(IX+d)
FDCB05F6	SET	6,(IY+d)
CBF7	SET	6,A
CBF0	SET	6,B
CBF1	SET	6,C
CBF2	SET	6,D
CBF3	SET	6,E
CBF4	SET	6,H
CBF5	SET	6,L
CBFE	SET	7,(HL)
DDCB05FE	SET	7,(IX+d)
FDCB05FE	SET	7,(IY+d)
CBFF	SET	7,A
CBF8	SET	7,B
CBF9	SET	7,C
CBFA	SET	7,D
CBFB	SET	7,E
CBFC	SET	7,H
CBFD	SET	7,L
CB26	SLA	(HL)
DDCB0526	SLA	(IX+d)
FDCB0526	SLA	(IY+d)
CB27	SLA	A
CB20	SLA	B
CB21	SLA	C
CB22	SLA	D
CB23	SLA	E
CB24	SLA	H
CB25	SLA	L
CB2E	SRA	(HL)
DDCB052E	SRA	(IX+d)
FDCB052E	SRA	(IY+d)
CB2F	SRA	A
CB28	SRA	B
CB29	SRA	C
CB2A	SRA	D

CODE OBJET	INSTRUCTION	
CB2B	SRA	E
CB2C	SRA	H
CB2D	SRA	L
CB3E	SRL	(HL)
DDCB053E	SRL	(IX+d)
FDCB053E	SRL	(IY+d)
CB3F	SRL	A
CB38	SRL	B
CB39	SRL	C
CB3A	SRL	D
CB3B	SRL	E
CB3C	SRL	H
CB3D	SRL	L
96	SUB	(HL)
DD9605	SUB	(IX+d)
FD9605	SUB	(IY+d)
97	SUB	A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
D620	SUB	n
AE	XOR	(HL)
DDAE05	XOR	(IX+d)
FDAE05	XOR	(IY+d)
AF	XOR	A
A8	XOR	B
A9	XOR	C
AA	XOR	D
AB	XOR	E
AC	XOR	H
AD	XOR	L
EE20	XOR	n

(Avec l'aimable autorisation de Zilog Inc.)

ANNEXE E
TABLE DE CONVERSION HEXADÉCIMALE

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	00	000
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	256	4096
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	512	8192
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	768	12288
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	1024	16384
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	1280	20480
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	1536	24576
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	1792	28672
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	2048	32768
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	2304	36864
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	2560	40960
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	2816	45056
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	3072	49152
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	3328	53248
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	3584	57344
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	3840	61440

5		4		3		2		1		0	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15

TABLE DES MATIÈRES

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par Rodnay Zaks*,
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par Rodnay Zaks*,
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par Rodnay Zaks*,
636 pages, Réf. 340

TECHNIQUES D'INTERFACE aux microprocesseurs *par Austin Lesea et Rodnay Zaks*,
450 pages, Réf. 339, 3ème édition

LEXIQUE INTERNATIONAL MICROORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par Joël Poncet*,
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par Pierre Le Beux*,
140 pages, Réf. 369

LA SOLUTION RS-232 *par Nelson Ford*,
208 pages, Réf. 0052

DYNAMIQUE CINÉMATIQUE, MÉTHODE POUR LA PROGRAMMATION DES JEUX *par Daniel Lebigre*,
272 pages, Réf. 0118

BASIC

VOTRE PREMIER PROGRAMME BASIC *par Rodnay Zaks*,
208 pages, Réf. 263

INTRODUCTION AU BASIC *par Pierre Le Beux*,
336 pages, Réf. 0035

LE BASIC PAR LA PRATIQUE : 60 exercices *par Jean-Pierre Lamoitier*,
252 pages, Réf. 0095

LE BASIC POUR L'ENTREPRISE *par Xuan Tung Bui*,
204 pages, Réf. 253, 2ème édition

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par Alan R. Miller*,
318 pages, Réf. 259

AU COEUR DES JEUX EN BASIC *par Richard Mateosian*,
352 pages, Réf. 233

JEUX D'ORDINATEUR EN BASIC *par David H. Ahl*,
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par David H. Ahl*,
204 pages, Réf. 247

FICHIERS EN BASIC *par Alan Simpson*,
256 pages, Réf. 0102

PASCAL

INTRODUCTION AU PASCAL *par Pierre Le Beux*,
496 pages, Réf. 330

LE PASCAL PAR LA PRATIQUE *par Pierre Le Beux et Henri Tavernier*,
562 pages, Réf. 361

LE GUIDE DU PASCAL *par Jacques Tiberghien*,
504 pages, Réf. 232

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par Alan R. Miller*,
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux*,
366 pages, Réf. 360

MICROORDINATEUR

ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut*,
96 pages, Réf. 320

ALICE et ALICE 90, PREMIERS PROGRAMMES *par Rodnay Zaks*,
248 pages, Réf. 376

ALICE, GUIDE DE L'UTILISATEUR *par Norbert Rimoux*,
208 pages, Réf. 378

ALICE, PROGRAMMATION EN ASSEMBLEUR *par Georges Fagot Barraly*,
192 pages, Réf. 420

AMSTRAD

AMSTRAD, PREMIERS PROGRAMMES *par Rodnay Zaks*,
248 pages, Réf. 0105

AMSTRAD, 56 PROGRAMMES *par Stanley R. Trost*,
160 pages, Réf. 0107

AMSTRAD, JEUX D'ACTION *par Pierre Monsaut*,
96 pages, Réf. 0108

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II, Tome 1 *par Léopold Laurent*,
208 pages, Réf. 333

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost*,
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE *par Douglas Hergert et Joseph T. Kalash*,
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert*,
272 pages, Réf. 0006

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR
par Joseph Caggiano,
208 pages, Réf. 396

APPLE IIC, GUIDE DE L'UTILISATEUR
par Thomas Blackadar,
160 pages, Réf. 0089

ATARI

JEUX EN BASIC SUR ATARI *par Paul Bunn,*
96 pages, Réf. 282

ATARI, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 387

ATARI, GUIDE DE L'UTILISATEUR *par Thomas Blackadar,*
192 pages, Réf. 354

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut,*
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost,*
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64
par Pierre Monsaut,
96 pages, Réf. 0017

COMMODORE 64, PREMIERS PROGRAMMES
par Rodnay Zaks,
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert,
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR
par J. Kassmer,
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES
par Stanley R. Trost,
192 pages, Réf. 319

COMMODORE 64, GUIDE DU GRAPHISME
par Charles Platt,
372 pages, Réf. 0053

COMMODORE 64, JEUX D'ACTION *par Eric Ravis,*
96 pages, Réf. 403

COMMODORE 64, 1ers CONTACTS
par Marty Dejonghe et Caroline Earhart,
208 pages, Réf. 390

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut,*
96 pages, Réf. 324

EXL 100

EXL 100, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0126

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL
par François Abella,
208 pages, Réf. 264

HECTOR

HECTOR JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 388

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier,*
256 pages, Réf. 338

IBM PC GUIDE DE L'UTILISATEUR
par Joan Lasselle et Carol Ramsey,
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 280

GRAPHIQUES SUR IBM PC *par Nelson Ford,*
320 pages, Réf. 357

LASER

LASER JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 371

MO 5

MO 5 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 0067

MO 5, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 370

MSX

MSX, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 411

MSX, INITIATION AU BASIC *par Rodnay Zaks,*
248 pages, Réf. 410

MSX, 56 PROGRAMMES *par Stanley R. Trost,*
160 pages, Réf. 0109

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir,
2 tomes, Réf. 261-262

SPECTRAVIDEO

SPECTRAVIDEO, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 377

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM
par S.M. Gee,
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4
par François Abella,
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 0026

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TO 7, PROGRAMMATION EN ASSEMBLEUR
par Georges Fagot Barraly,
192 pages, Réf. 350

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80
par Léopold Laurent,
2 tomes, Réf. 250-251

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2
par Michel Lhoir,
2 tomes, Réf. 366-262

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR
par Pierre Monsaut,
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR
par Orson Kellog,
112 pages, Réf. 300

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20
par G. O. Hamann,
2 tomes, Réf. 244-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

GUIDE DU BASIC VIC 20, COMMODORE 64
par Douglas Hergert,
240 pages, Réf. 312

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

VG 5000

VG 5000, JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 422

VG 5000, 56 PROGRAMMES *par Stanley R. Trost*
160 pages, Réf. 0128

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 256

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

MICROPROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 220

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 274

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 0031, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

PROGRAMMATION DU 6800
par Daniel Jean David et Rodnay Zaks,
374 pages, Réf. 327

PROGRAMMATION DU 6809
par Rodnay Zaks et William Labiak,
392 pages, Réf. 0139

PROGRAMMATION DU 8086/8088
par James W. Coffron,
304 pages, Réf. 0016

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 0133

ASSEMBLEUR DU 8086/8088
par François Retaureau,
616 pages, Réf. 0093

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD
par Charles W. Grant et Jon Butah,
308 pages, Réf. 365

GUIDE DU PC DOS *par Richard A. King*,
240 pages, Réf. 0013

GUIDE DE MS-DOS *par Richard A. King*,
360 pages, Réf. 0117

LOGICIELS ET APPLICATIONS

INTRODUCTION AU TRAITEMENT DE TEXTE
par Hal Glatzer,
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman*,
200 pages, Réf. 0062

WORDSTAR APPLICATIONS *par Julie Anne Arca*,
320 pages, Réf. 0005

VISICALC APPLICATIONS *par Stanley R. Trost*,
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par Dominique Helle*,
304 pages, Réf. 309

INTRODUCTION A dBASE II *par Alan Simpson*,
280 pages, Réf. 0064

DE VISICALC A VISI ON *par Jacques Bourdeu*,
256 pages, Réf. 321

MULTIPLAN POUR L'ENTREPRISE
par D. Helle et G. Boussand,
304 pages, Réf. 0079

dBASE II APPLICATIONS *par Christophe Stehly*,
248 pages, Réf. 416

MULTIPLAN SUR MACINTOSH
par Goulven Habasque,
240 pages, Réf. 0099

INTRODUCTION A LOTUS 1-2-3
par Chris Gilbert et Laurie Williams,
272 pages, Réf. 0106

La plupart de ces ouvrages existent en version anglaise. N'hésitez pas à demander notre catalogue.

EN ANGLAIS

BASIC EXERCISES FOR APPLE *by Jean Pierre Lamoitier*,
232 pages, Réf. 0-084

BASIC FOR BUSINESS *by Douglas Hergert*,
224 pages, Réf. 0-080

CELESTIAL BASIC : Astronomy on your Computer
by Eric Burgess,
228 pages, Réf. 0-087

INTRODUCTION TO PASCAL (Including UCSD Pascal)
by Rodnay Zaks,
422 pages, Réf. 0-066

DOING BUSINESS WITH PASCAL
by Richard Hergert and Douglas Hergert,
380 pages, Réf. 0-091

MASTERING VISICALC *by Douglas Hergert*,
224 pages, Réf. 0-090

THE APPLE CONNECTION *by James W. Coffron*,
228 pages, Réf. 0-085

PROGRAMMING THE Z8000 *by Richard Mateosian*,
300 pages, Réf. 0-032

A MICROPROGRAMMED APL IMPLEMENTATION
by Rodnay Zaks,
350 pages, Réf. 0-005

ADVANCED 6502 PROGRAMMING *by Rodnay Zaks*,
292 pages, Réf. 0-089

FORTRAN PROGRAMS FOR SCIENTISTS AND ENGINEERS *by Alan R. Miller*,
320 pages, Réf. 0-082

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 203.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 626441
Telex : 08588163

ANGLETERRE
Unit 4 - Bourne Industrial Park
Bourne Road, Crayford
Kent DA1 4BU
Tel. : (0322) 57717



Paris • Berkeley • Düsseldorf • Londres

Achevé d'imprimer le 15 mai 1985 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 2e trimestre 1985 - Imprimeur n° 1774

Tous les utilisateurs d'Amstrad qui veulent aller plus loin après avoir maîtrisé les ressources du BASIC trouveront dans cet ouvrage les éléments nécessaires pour aborder la programmation en assembleur. Après un court rappel d'arithmétique binaire, les principales instructions du microprocesseur Z 80 sont décrites et accompagnées d'exemples de sous-programmes. Le lecteur apprendra ainsi comment réaliser des programmes beaucoup plus performants tout en comprenant mieux le fonctionnement de son micro-ordinateur.



L'AUTEUR

GEORGES FAGOT BARRALY est professeur de mathématiques et spécialiste de l'utilisation des micro-ordinateurs dans l'enseignement. Il a réalisé de nombreux programmes didactiques et il est également l'auteur de plusieurs livres sur la programmation en assembleur.




SYBEX
AMSTRAD PROGRAMMATION EN ASSEMBLEUR



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/